DOCTOR THESIS

Genetic Algorithm-based Optimization of Generative Adversarial Networks and its Applications

HE Bate
Department of Complex Systems Science
Graduate School of Informatics
Nagoya University
Japan

Contents

Abstract

The time-series data is a series of values obtained by continuously observing changes over time in a specific phenomenon; such as economic data, weather changes data, airplane ticket prices and so on. Since the accurate prediction of the time-series data is vital for everyday life, several algorithms are studied widely by many researchers. Previous time-series analysis includes Autoregressive conditional heteroscedasticity model (ARCH), Autoregressive integrated moving average (ARIMA), Markov model, and machine learning algorithms (Recurrent Neural Networks, Long Term-Short Memory, etc.). Although, recently, supervised machine learning algorithms have been successively applied to predict the time-series data, the training process needs a large amount of learning data to develop the mathematical model of the prediction. Therefore, unsupervised learning is advantageous in analyzing and predicting time-series data because it can use less unlabeled training data. As a representative of unsupervised learning, Generative Adversarial Networks (GANs) have an outstanding ability to learn data distribution and generate data tasks not only in the image data domain, but also in the text data, and other non-image data domain. In this study, GANs model is applied for time-series data prediction.

Some researchers have been applied GANs for the prediction of the time-series data and their researches show that the prediction model based on GANs has better results than previous methods. However, there is one problem to be solved. The prediction accuracy of the GANs-based model depends on the hyper-parameters of the GANs and the data preprocessing for the prediction model. In the previous studies, they were determined manually by humans. Since the design of hyper-parameters and the data pre-processing depend on the different GANs model and the task to be solved, it is difficult for inexperienced researchers to determine them in advance. Design of the hyper-parameters of the GANs and the data pre-processing by Genetic Algorithm (GA) is presented in this study. GA is widely used in optimizing deep learning frameworks such as Convolutional Neural Networks and has achieved excellent results. For this reason, GA is used to optimize GANs, which have a more complex structure. For an optimization problem, GA represents a certain number of candidate solutions as chromosomes so that the population can evolve to a better solution.

Therefore, the purpose of this research is as follows:

1. The first is to establish a suitable Generative Adversarial Networks (GANs) model for time-series numerical data and continuous images data in order to maximize the potential ability of GANs for predicting time-series data.

2. The second is to present the way to design the hyper-parameters of the GANs automatically and the data pre-processing by Genetic Algorithm (GA).

The validity of the proposed model is discussed in the experiments on stock data prediction and continuous walking images of pedestrians.

GANs have two networks: a Generator and a Discriminator. In the application of stock data prediction, the adequate selection of the networks for a Generator and a Discriminator is discussed firstly. Long Short-Term Memory (LSTM) is adopted for the Generator, and then, Multi-layer perceptron (MLP), Recurrent Neural Network (RNN), Gated Recurrent Unit (GRU), and Long Short-Term Memory (LSTM) are compared for the Discriminator. The results show that the best accuracy is observed when Long Short-Term Memory (LSTM) is used for both the Generator and the Discriminator. A GA-based optimization algorithm is applied to optimize both sliding window size for data pre-processing and hyper-parameters for GANs, such as the number of hidden layers and the number of units on each hidden layer. The results show that the proposed algorithm can simultaneously determine the sliding window size for the proposed model and the hyper-parameters. Besides, the prediction accuracy has been improved.

In the successive image generation of pedestrian walking behavior, GANs model adopts convolutional neural networks (CNN) as the Generator and the Discriminator. Then, the continuous past images are taken as input data for GANs, and then, the continuous future images are generated. Another GA-based optimization algorithm is applied for the optimization of both number of input and output images and the hyper-parameters of CNN such as the number of hidden layers and activation functions. The results show that the proposed algorithm can simultaneously determine the number of input and output images and hyper-parameters. Besides, the quality of the generated image has been improved.

In Automated Machine Learning (AutoML), researchers tend to focus on the model construction for machine learning automation and ignore the importance of data pre-processing.

This study simultaneously optimizes data pre-processing and model construction, breaking through the previous limitation that AutoML only optimizes the model network structure.

Chapter 1

Introduction

1.1 Research Background

Artificial intelligence is a new technical science that uses and develops theories, methods, technologies, and application systems to simulate, extend and expand human intelligence. The purpose of the research is to promote intelligent machines for speech recognition, machine translation, image recognition, text recognition, speech synthesis, human-computer dialogue, human-computer games, theorem proofs, robots, and autonomous driving cars, for example. In the summer of 1956, scientists such as McCarthy and Minsky [1] held a meeting at Dartmouth College in the United States to discuss "how to use machines to simulate human intelligence." This meeting marked the birth of the intelligent discipline.

In this thesis, the development of artificial intelligence is divided into the following six stages:

The first stage is the initial development period: 1956 to the early 1960s. After the concept of artificial intelligence was put forward, several remarkable research [2] results have been achieved, such as machine theorem proofs, checkers programs, etc., which set off the first examples of the development of artificial intelligence.

The second is to reflect on the development period: 1960s-early 1970s. The breakthrough progress in the early stage of the development of artificial intelligence has significantly increased people's expectations of artificial intelligence [3] [4]. People began to try more challenging tasks, putting forward unrealistic research and development goals. However, successive failures and the failure of expected goals (for example, the inability to use machines to prove whether the sum of two continuous functions is a continuous function) have not

helped the jokes about machine translation.

The third stage is the application development period: the early 1970s to the mid-1980s [5]. The expert system that emerged in the 1970s simulated the knowledge and experience of human experts to solve problems in a specific field, and a breakthrough in artificial intelligence from theoretical research to the practical application was realized, from the discussion of general reasoning strategies to the use of specialized knowledge.

The fourth stage is the period of downturn development: the mid-1980s to the mid-1990s [6]. As the application scale of artificial intelligence continued to expand, problems such as a lack of application fields, lack of common-sense knowledge, difficulty in acquiring knowledge, single reasoning methods, lack of distributed functions, and difficulty in compatibility with existing databases were gradually exposed.

The fifth is a period of steady development: the mid-1990s to 2010. Due to the development of network technology, especially internet technology, the innovation research of artificial intelligence was accelerated, and artificial intelligence technology moved further towards practicality. In 1997, the International Business Machines Corporation (IBM) Deep Blue supercomputer [7] defeated the world chess champion Kasparov. In 2008, IBM put forward the concept of "Smart Earth". The above are all landmark events of artificial intelligence in this period.

The sixth stage is the period of vigorous development: 2011 to present. With the improvement of both software (machine learning algorithms) and hardware (CPU, GPU, cloud computing) in computer science, machine learning has been overloaded from theoretical algorithms to practical applications. The applications include various application cases such as image classification [8–11], speech recognition, question-answer system, and automatic drive. This development has led to a new surge of explosive growth in machine learning.

Machine learning can be divided into the following categories:

- Supervised learning [12] learns a function from a given training data set. When new data arrives, it can predict the results based on this function. The training set requirements for supervised learning include input and output, which can also be said to be features and targets. Standard supervised learning algorithms include regression

analysis and statistical classification.

- Compared with supervised learning, unsupervised learning [13] has no artificially labeled results in the training set. Standard unsupervised learning algorithms include Generative Adversarial Network (GANs) [14] and clustering.

- Semi-supervised learning [15] is somewhere between supervised learning and unsupervised learning.

- Reinforcement Learning (RL) [16] is a field of machine learning that learns "what to do (i.e., how to map the current situation to an action) to maximize the numerical earnings." The learner is not told what actions to take but must try to discover for himself which actions will yield the most rewarding results.

The difference between supervised learning and unsupervised learning is whether researchers label the target of the training set. Researchers use training sets, and they all have inputs and outputs.

Machine learning is represented by supervised learning Convolutional Neural Networks (CNN) [17] and has achieved great success. However, CNN is not good at data generation and fitting data distribution, and the emergence of GANs provides a good idea and framework for this field. At present, machine learning is divided into four aspects: supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning. CNN has shown compelling performance in the field of supervised learning. The emergence of GANs has taken the field of unsupervised learning a big step forward. GANs are composed of the Generator and Discriminator. The advantages of this framework are as follows:

- The model only uses back-propagation, without a Markov chain [18].

- In theory, if a function is differentiable, it can be used to construct Discriminator and Generator because it can be combined with a deep neural network to make a deep generative model.

- The Generator's parameter update is not directly from the data sample but instead uses the back-propagation from the Discriminator (this is the most different from the traditional method).

1.2 Issues and Challenges

Although GANs have made significant breakthroughs in generating data [19–30], and have also achieved specific results in time-series prediction [31–36], GANs still have some unsolved problems in the prediction of time-series data. This research focuses mainly on the following unsolved problems:

1. Constructing the corresponding GANs model based on different time-series data is the first problem. GANs are not so much a model as a framework, in which the Generator and Discriminator are essential components. There are complex problems for training GANs when choosing the differentiable network structure, setting the corresponding loss function, and setting the input and output. Since the input of the original GANs is a random vector, it will be difficult to train if the random vector is also used when processing the time-series data. Besides, time-series data is generally unstructured data. Although GANs do not require a large amount of manually labeled structured data compared to traditional CNN, a certain degree of data pre-processing is necessary.

2. Optimization of models based on GANs. When using GANs-based models to train and predict time-series data, the GANs model has to be optimized to achieve the best results. In the supervised learning represented by CNN, many optimization methods have been produced, including the multi-objective optimization method [37, 38], the ensemble method [39–42], and the genetic algorithm-based method [43–46], for example, but the optimization of GANs is focused on improving the quality of the generated pictures [47–55]. There are relatively few optimization methods for time-series data. Since GANs are composed of two differentiable networks against training, it is almost impossible to optimize two networks simultaneously, so finding a way to optimize GANs to improve their expressiveness is an important topic.

These two issues are challenging for researchers. In particular, in the second problem above, optimizing GANs-based models can not only promote the automatic generation of

models, but also automatically make different models for different datasets, which makes the GANs-based model more flexible in solving problems. Predicting future data based on historical time-series data can be applied in many fields, and solving these problems can better help these applications.

1.3 Related Works

Works related to these issues in this thesis are summarized below.

1.3.1 Stock price prediction

The prediction of time-series data, such as in economic data, weather changes, and airplane ticket prices, for example, has important significance to everyday life. The changes in these data are also based on changes in historical data. Conventional and state-of-art methods for time-series data prediction will be introduced in this section.

In the area of Quantitative Finance, the Autoregressive Integrated Moving Average model (ARIMA), one of the methods of time-series forecast analysis [56–59], was widely used before neural networks. The ARIMA model has three parameters (p, d, q), which represent the order of the autoregressive model, the degree of differencing, and the order of the moving-average model, respectively. Although the ARIMA model is simple enough only to need endogenous variables and not need other exogenous variables, it is a linear model which and has two serious demerits:

- The time-series data are required to be stationary, or it must be stable after being differencing.
- Essentially, the ARIMA model can only capture linear relationships in a time-series dataset and not in nonlinear relationships.

Long Short-Term Memory (LSTM) [60] is one type of Recurrent Neural Network (RNN) [61] architecture. Unlike ARIMA, LSTM can mine the nonlinear information from time-series data like stock market data [62–66]. At first, LSTM was used for Natural Language Processing (NLP). Currently, it can be used with speech recognition tasks and time-series tasks for capturing deep patterns from these datasets [67–69]. In the study, "Stock market's

price movement prediction with LSTM neural networks" [62], the researchers used LSTM networks to predict future trends of stock prices based on the historical stock price. The researchers achieved 55.9% accuracy on average whether the stock was forecast to go up or not. In the research "Applying long short term memory neural networks for predicting stock closing price" [63], the researchers used stock data from the S&P and the NASDAQ to predict next-day stock movement by using the LSTM model. The results showed that LSTM had a slightly higher prediction accuracy than conventional methods, such as ARIMA.

Recently, GANs have also been applied to stock prediction [70]. In the research "Stock market prediction on high-frequency data using generative adversarial nets" [71] the researcher employed LSTM and CNN for adversarial training to forecast high-frequency stock data. Their experiments showed that their approach effectively improves stock price prediction accuracy and reduces forecasting errors. In "Stock market prediction based on generative adversarial network" [72], the researchers also used GANs as their basic framework, and they set MLP as the Discriminator and LSTM as the Generator.

In addition to LSTM and GANs, researchers have also used other neural network methods to predict stock movements [73–78].

The above researches used GANs for stock prediction and achieved some promising results. However, the models have not been optimized, and it is not clear which model can perform best. In other words, these models lack good interpretability.

1.3.2 Successive Image Generation

The pedestrian prediction method will be introduced in this section. Although this research focuses on the successive image generation of pedestrians [79–81], there are some other prediction methods for pedestrian prediction task [82–86].

In the research "Planning-based prediction for pedestrians." [87] the researchers presented an approach that can determine robot movements while not hindering the movements of people to replace pedestrian's trajectories. The experiments were held in an office environment.

In work "Deep multi-scale video prediction beyond mean square error" [88], the authors learned to predict future images from a video (successive images). The authors trained a

convolutional network to generate future images that were given an input sequence. In addition, the authors used Mean Squared Error (MSE) as the loss function. In another similar work, "Generating videos with scene dynamics" [89], the researchers also used GANs for the video generation tasks. They separated the successive images (video data) into foreground and background. Their experiments showed tiny videos could be generated up to a second, which is better than simple baselines. However, the results cannot generate tiny videos logically and showed video images such as humans having three eyes. In the research, "To Create What You Tell: Generating Videos from Captions" [90], the authors tried to generate videos from an input sentence that contained the movement of objects such as an image including digit 6 moving up and down.

Since researchers can obtain helpful information by predicting the future of frames of the video, and GANs have shown advantages over traditional methods in the above several studies, the results from the generated videos are often incomprehensible to humans. These incomprehensible results occur for reasons: first, too many categories of the data set make the GANs model unable to learn too many video features; second, the researchers did not optimize when building the model. As a result, the model did not perform at its best.

1.3.3 Others time-series data generation by using GANs

There are also other applications using GANs on time-series data prediction. In the research, "C-rnn-gan: Continuous recurrent neural networks with adversarial training" [31], the authors proposed a GANs model that uses continuous sequential data, which is a collection of classical music. They concluded that the model could generate great music as judged by the audience. In addition, in research entitled, "Real-valued (Medical) Time Series Generation with Recurrent Conditional GANs" [32], the researchers proposed a Recurrent GAN (RGAN) and Recurrent Conditional GAN (RCGAN) produce realistic medical data.

One problem with GANs is that it has limitations when the purpose is for generating sequences for discrete data. To overcome this problem, SeqGAN is proposed in the study "SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient" [91], where the generator is an intelligent body in reinforcement learning, the state is the generated token, the action is the next generated token, a Monte Carlo search is used to estimate the state behav-

ior value, and the policy gradient is used to train the gradient. The discriminator is a CNN network that evaluates the generated sequences and guides the generative model learning.

A good generative model preserves the characteristic of a temporal dynamic. In other words, the generated sequence can restore the relationship between each time step of the original. In the research, "Time-series Generative Adversarial Networks" [92], the authors discuss how the previous proposed GANs methods do not sufficiently preserve the temporal correlation of time series. A new GANs framework is proposed in this work. It combines the flexibility of the unsupervised paradigm with supervised training. By combining a learning embedding space jointly optimized for supervised and adversarial objectives, the network is allowed to comply with dynamic soft-constrained data during sampling.

1.3.4 Optimization of GANs

Automated Machine Learning (AutoML) has become a trend in the development of machine learning since CNN has become widely used. AutoML can not only make machine learning models achieve the best pattern recognition results but can also automatically build corresponding models for different data sets. Most importantly, this process is automated and does not require the machine learning experience of experts to design unique models.

The Genetic Algorithm used in this study to optimize GANs has been used to optimize CNN automatically in previous studies [93–96]. To search for neural network architecture, the paper "Genetic CNN" [97] limits the network to a limited depth, and each layer is a preset operation, but there are still many candidate networks. To search effectively in a vast search space, "Genetic CNN" proposes accelerating a Genetic Algorithm. In order to do this, the authors first construct the initial population, then perform genetic selection operations, such as crossover and mutation on the individuals in the population, judge their adaptability through the accuracy of recognition, and finally obtain a substantial population. Such an encoding form can encode the current mainstream classification structure, but it also has many limitations: 1) The current connection method requires only convolution and pooling, and other tricky modules, such as Maxout, cannot be used. 2) The convolution kernel at each stage is fixed, which hinders the fusion of multi-scale features. In another work, "Automatically Designing CNN Architectures Using Genetic Algorithm for Image Classifica-

tion" [98], the researchers also proposed an automatic CNN architecture design approach using a Genetic Algorithm.

Many works on optimizing CNN structure using a Genetic Algorithm exist, but few works on optimizing GANs exist. In work "Evolutionary generative adversarial networks" [99], the authors convert the adversarial training process of the original GANs into an evolutionary problem, making the entire training process more stable and improving the performance of the generator. This optimization process has three operations:

- Variation: The link of the variation is the process of generating offspring $G_{\theta 1} G_{\theta 2} ... G_{\theta n}$ through an individual G_θ through a mutation operator
- Evaluation: For each generated offspring to be evaluated by a fitness function, the fitness function must depend on the current environment (the discriminator)
- Selection: Excellent offspring should be selected according to the fitness value to enter the next evolutionary iteration process

The crossover operation did not appear in the whole optimization operation, and the chromosome did not emerge for iteration. Therefore, in this research, a Genetic Algorithm cannot be accurately used, but an algorithm that dynamically changes the loss function is used instead to optimize GANs.

In work, "Genetic algorithm-optimized long short-term memory network for stock market prediction" [100], the researchers used a Genetic Algorithm to optimize a long short-term memory network for stock market prediction. They set the LSTM's inside structure as a hyper-parameter which had to be optimized.

1.4 Motivations and Contributions

The research above has contributed to GANs breakthroughs in modeling and predicting time-series data that were not apparent in previous single models, but there is still much room for improvement in prediction accuracy and optimization. Related studies such as E-GAN, which is the first application of a genetic algorithm to GANs, however, generate higher quality images. E-GAN is not considered a genetic algorithm in the strict sense because it

does not use crossover and variation. The main contribution of this study is the optimization of GANs when processing time-series data using genetic algorithms. Previous research has focused on optimizing the structure of the machine learning model itself, but while the model itself is crucial in machine learning, processes such as data pre-processing can also have a massive impact on the training results. This study uses genetic algorithms to combine the data pre-processing and the training model for optimization. This has the advantage of eliminating the need for the experience and knowledge of human experts in adjusting hyper-parameters when processing time-series data for prediction tasks with GANs models and making the entire training process more automated. This study uses these advantages as innovative points for experiments, and the time-series data are chosen for two reasons:

- Stock data is chosen among many time-series data for the following reasons: stock data has a relatively high value for analysis, and due to the rise of machine learning, there are more and more studies applying machine learning to economic analysis; secondly, stock data is relatively easy to obtain, through some packages such as the mlpfinance library of python, for example. Various types of stock data can be obtained directly through this and other packages. Also, stock data are somewhat cyclical and can be modeled by historical data to make predictions.

- The image data of a pedestrian walking continuously was chosen because we think the GANs handle time-series data differently when facing numerical data and successive image data. If the following image of the pedestrian can be predicted, then the application scenario will be extensive.

The main contributions of this study are as follows.

1. This study proposes a model for analyzing and predicting time-series data based on GANs. It aims to go from unsupervised to semi-supervised learning by changing the way the data are trained.

2. This study proposes a genetic algorithm-based optimization method that automatically generates an optimal GANs model for each time-series data. This optimal model obtained can maximize the predictive power of the model.

3. This study combines the data pre-processing process and the process of GANs model building for optimization, which extends the machine learning automation extended.

1.5 Overview of the Thesis

This paper consists of seven chapters.

Chapter 1 includes the research background, its issues and challenges, and a discussion of related research in four categories: 1) Stock price prediction, 2) Successive Image Generation, 3) Other time-series data generation using GANs, and 4) Optimization on GANs. Chapter 1 also concludes with the motivations and contributions and the thesis.

Chapter 2 describes the fundamental methodologies that are used in the proposed methods and several comparative algorithms. Generative Adversarial Networks are highlighted because these Networks are the basic framework used in this thesis. Convolutional Neural Networks are introduced for use with successive images. Recurrent Neural Networks, Long Short-Term Memory, and Gated Recurrent Units are introduced for use with the time-series stock data. A Genetic Algorithm was introduced for optimization on sequential GANs. Comparative Algorithm, which includes ARIMA, and optical flow, is also introduced.

Chapters 3 and 4 describe the first application task, i.e., stock prediction. Chapter 3 describes how the GANs model, the pre-processing data method, and the corresponding training strategy are built for the stock data, the time-series data which is used in this study, for the prediction task. Furthermore, it compares with the previous research methods. The original GANs are first converted into GANs that can process time-series data. LSTM is set up in the generator because LSTM performs best in the distribution of time-series data when capturing. Suitable for the Discriminator, MLP, RNN, LSTM, GRU are set up. Two training methods are proposed: sliding windows, segmented training. Experiments prove that the combination of G-LSTM and D-LSTM performs best. Chapter 4 optimizes the GANs model with the best results using genetic algorithm based on Chapter 3. The GANs model and data pre-processing are combined and optimized simultaneously for machine learning automation. The input data and the hyper-parameters in the generator's network structure are combined for training. The results show that the optimized GANs model is better than the unoptimized model. The performance is even better, although this adds much time to

training the model.

Chapters 5 and 6 describe the second application task, i.e., successive image generation. In this application task, video data of walking pedestrians, in other words, successive image data, is used for analysis and prediction. Since successive image data is somewhat different from continuous time-series data, the study's modeling details and optimization modeling process of GANs are different from Chapters 3 and 4. In the model establishing process, reverse CNN are set to the generator in GANs, and CNN (like a two-classifier) are set to the Discriminator. Then, RMSE and a CNN 8-classifier are used to evaluate the results of generated future images by using GANs. Experiments show that the results of our proposed method are better than previous studies. The genetic algorithms are used to establish an optimization scheme based on this model. The number of input pictures, the number of output (predicted) pictures, and the hyper-parameters in the generator are combined with the network structure for training. The results show that the optimized GANs model performs better than the unoptimized model.

Chapter 7 concludes this research. The conclusions are summarized, and the overall contribution of this work is discussed as well as suggestions for future work.

Chapter 2

Theory

2.1 Multi-layer Perceptron

A Multi-layer Perceptron (MLP) [101] is also called an Artificial Neural Network (ANN). A neural network is a simulation and simplification of biological neurons. Biological neurons are composed of dendrites, cell bodies, axons, and other parts. Dendrites are the input end of the cell body and receive surrounding nerve impulses; axons are the output end of the cell body, transmitting nerve impulses to other neurons. Biological neurons have two states of excitement and inhibition. When the receiving stimulus is higher than a certain threshold, it will enter an excited state and transmit the nerve impulse from the axon; otherwise, there is no nerve impulse. The basic structure of the multi-layer perceptron (MLP) is based on the biological neuron model. The most typical MLP includes the input layer, hidden layer, and output layer. The different layers of the MLP neural network are fully connected (fully connected means that any neuron in the upper layer is connected to all neurons in the next layer), as shown in Figure 2.1.

If the input has N dimensions, the input layer has N neurons.

The hidden layers are fully connected with the input layer. If the input layer is represented by X, the output of hidden layers is $f(W_1X + b_1)$, where W_1 is weight, b_1 is the bias, and the function f can be a sigmoid function or tanh function.

The hidden layers to the output layer can be regarded as a multi-category logistic regression, a softmax regression. Therefore, the output of output layer is $softmax(W_2X_1 + b_2)$, where X_1 represents the output of hidden layers $f(W_1X + b_1)$.

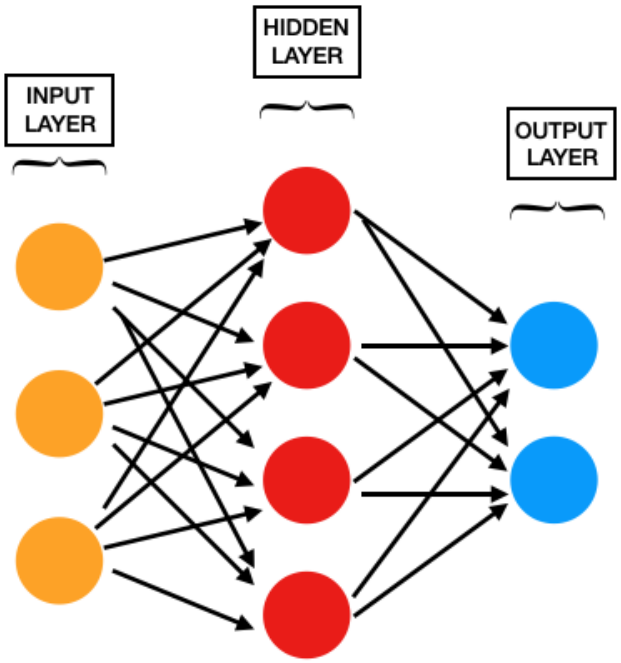

Fig 2.1: Multi-layer Perceptron

In summary, MLP can be expressed as the following function:

$$f(x) = G(b_2 + W_2(sigmoid(b_1 + W_1 x))) \tag{2.1}$$

Therefore, all the parameters of MLP are the connection weights and biases between layers, including W_1, b_1, W_2 and b_2. For a specific problem, solving the best parameter is an optimization problem. To solve the optimization problem, the simplest method is using the Gradient Descent [102]. First, all parameters are initialized randomly, then iteratively trained, and the training process continuously calculates the gradient and updates the parameters until a certain value is satisfied.

2.2 Convolutional Neural Network (CNN)

Like the way children learn to recognize objects, an algorithm is needed that shows millions of pictures before the input can be generated and predictions made about images that have never been seen before.

Computers "see" things in different ways than we do. Their world consists only of num-

bers. Each image can be represented as a two-dimensional array of numbers, called pixels.

However, the fact that they perceive images differently does not mean that we cannot train their recognition patterns, just like how we do to recognize images. We need to think about what an image is differently.

To "teach" an algorithm to recognize objects in an image, a specific type of artificial neural network has to be used: a Convolutional Neural Network (CNN). The name comes from one of the most critical operations in the network: convolution.

Convolutional neural networks are inspired by the brain. DH Hubel and TN Wiesel's research [103] on the mammalian brain in the 1950s and 1960s proposed a new model of how mammals perceive the world visually. They showed that the visual cortex of cats and monkeys includes neurons that specifically respond to neurons in their immediate environment.

In their paper, the authors described two basic types of visual neuron cells in the brain, each of which functions in a different way: simple cells (S cells) and composite cells (C cells).

In 1980, a researcher named Fukushima proposed a hierarchical neural network model [104]. He called it the new cognition. The model was inspired by the concept of simple and complex cells. Fukushima described a 'neocognitron', which can recognize patterns by knowing the shape of objects.

Later, in 1998, the convolutional neural network was introduced by Bengio, Le Cun, Bottou, and Haffner [105]. Their first convolutional neural network is called LeNet-5, which can classify numbers in handwritten digits.

A Convolutional Neural Network (Convolutional Neural Network) is abbreviated as CNN. CNN is mighty in image recognition. Many image recognition models have been extended based on the CNN architecture. It is also worth mentioning that the CNN model is one of the few deep learning models built regarding the visual organization of the human brain.

Figure 2.2 depicts a CNN model for pattern recognition. The image at the leftmost is our input layer, which the computer understands as inputting several matrices, basically in the same way as DNN.

The input image is followed by the Convolution Layer (Convolution Layer), which is unique to CNN. The activation function of the convolutional layer uses ReLU. It is very simple, namely, ReLU(x)=max(0,x). Behind the convolutional layer is the Pooling layer, which is

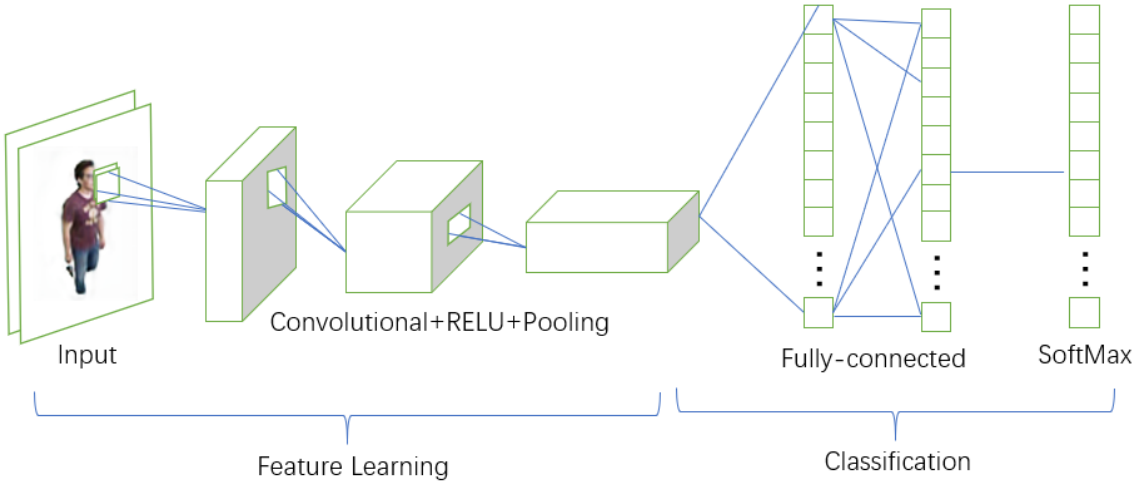

Fig 2.2: Convolutional Neural Network

also unique to CNN. It should be noted that there is no activation function for the pooling layer.

The combination of the convolutional layer + the pooling layer can appear many times in the hidden layer, and it appears twice in the above figure. This number is based on the needs of the model. Of course, the convolutional layer + convolutional layer can be used flexibly, or the combination of convolutional layer + convolutional layer + pooling layer, so that there are no restrictions when building the model. However, the most common CNN is a combination of several convolutional layers + pooling layers, such as the CNN structure in the figure above.

Behind several convolutional layers+pooling layers is a fully connected layer (Fully Connected Layer, referred to as FC). The fully connected layer is the DNN structure, but the output layer uses the Softmax activation function for image recognition classification.

From the above CNN model description, compared to DNN, CNN is more remarkable in the convolutional layer and pooling layer.

2.2.1 Convolutional layer

For image convolution, different local matrices of the input image and the elements in each position of the convolution kernel matrix are multiplied, and then added.

In the following example, the input in the figure is a two-dimensional 3x4 matrix, and

the convolution kernel is a 2x2 matrix. In this case, the convolution is convolved by moving one pixel at a time so that the upper-left corner of the input 2x2 is convolved with the convolution kernel, that is, the elements at each position are multiplied and then added, and the output matrix S is the value, which is aw+bx+ey+fz. Next, the input part is translated to the right by one pixel, and then it is a matrix composed of four elements (b,c,f,g) and the convolution kernel to convolve so that the elements of S01 of the output matrix S result. It uses the same method, elements of S02, S10, S11, S12, S02, S10, S11, S12 of the output matrix S result.

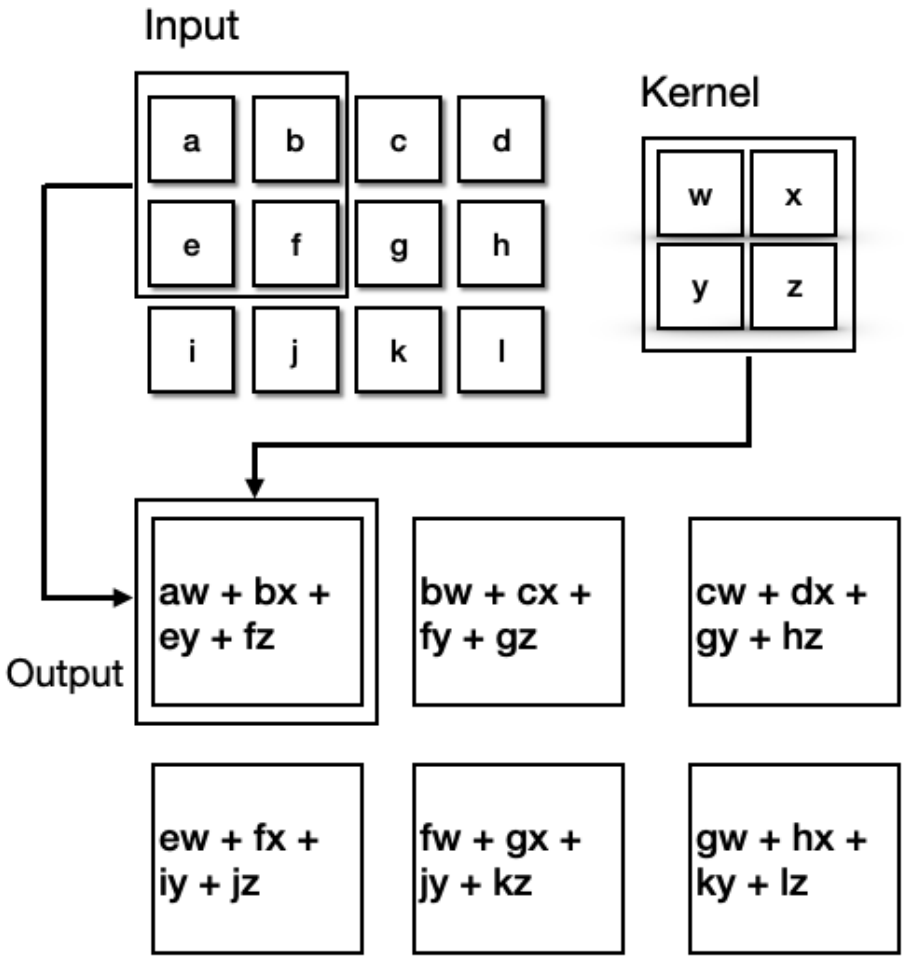

Fig 2.3: Convolution layer

2.2.2 Activation Function

Each neuron node in the neural network accepts the output value of the previous layer of a neuron as the input value of this neuron and passes the input value to the next layer. The input layer neuron node will directly pass the input attribute value to the next layer (hidden layer or output layer). In a multi-layer neural network, there is a functional relationship between the upper node's output and the lower node's input. This function is called the activation function.

If the activation function has not been used, the activation function is f(x) = x, and the input of each layer of the node is the linear function of the output of the upper layer. Regardless of the many layers of a neural network, the output is a linear combination of the input, which is equivalent to the effect of no hidden layer. In this case, the model is the most primitive perceptron (MLP), so the approximation ability of the network is quite limited. Because of the above reasons, a non-linear function will be used as the excitation function so that the deep neural network expression ability is more powerful (no longer a linear combination of inputs, but almost any function can be approximated).

Sigmoid is a commonly used nonlinear activation function, and its mathematical form is as follows:

$$f(z) = \frac{1}{1+e^{-z}} \tag{2.2}$$

The geometric image of Sigmoid is as follows:

The sigmoid function can transform the continuous real value of the input into an output between 0 and 1. In particular, if the input is a vast negative number, the output is 0; if the input is a vast positive number, the output is 1.

However, the sigmoid function has some flaws:

- Disadvantage 1: In the deep neural network, it causes gradient explosion and gradient disappearance when the gradient is transferred backward. The probability of gradient explosion is minimal, and the probability of gradient disappearance is relatively large.

- Disadvantage 2: Sigmoid function's output is not zero mean (i.e., zero-centered). This is undesirable because it will cause neurons in the next layer to get the non-zero mean

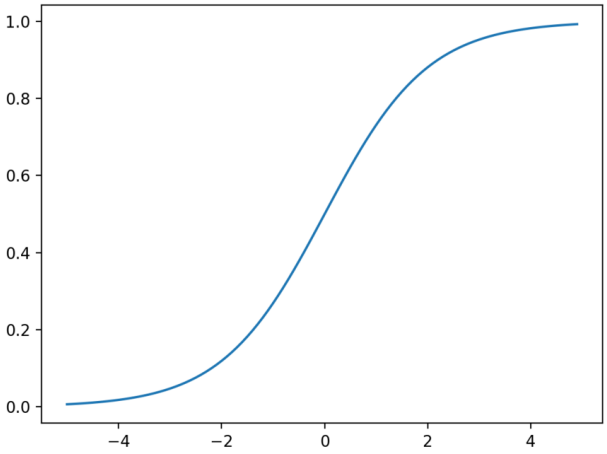

Fig 2.4: Sigmoid function

output signal of the previous layer as input.

- Disadvantage 3: Sigmoid function's analytical formula contains exponentiation, which is relatively time-consuming to solve by computer. For large-scale deep networks, this will greatly increase the training time.

The Tanh function's mathematical form is as follows:

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.3}$$

The geometric image of Tanh is as follows:

The Tanh function [106] is omitted by a Hyperbolic Tangent. The Tanh function solves the problem of a non-zero-centered output of the Sigmoid function. However, the problems of gradient vanishing and exponentiation still exist.

The ReLU function's mathematical form is as follows:

$$Relu = max(0, x) \tag{2.4}$$

The geometric image of ReLU is as follows:

The ReLU function [107] is a maximum value function. Note that this is not all-interval derivable, but the sub-gradient, as shown in the figure above, can be used. Although ReLU

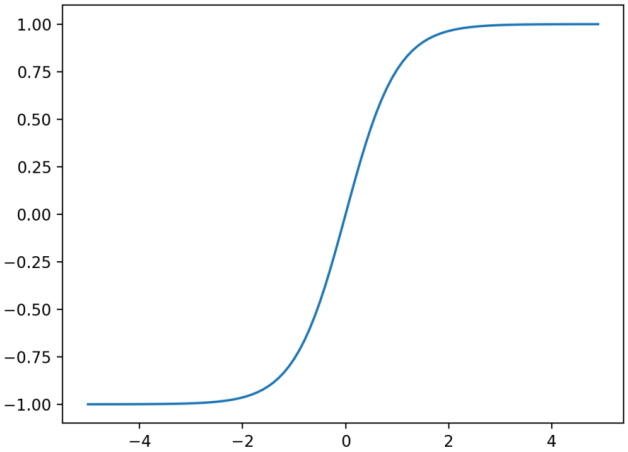

Fig 2.5: Tanh function

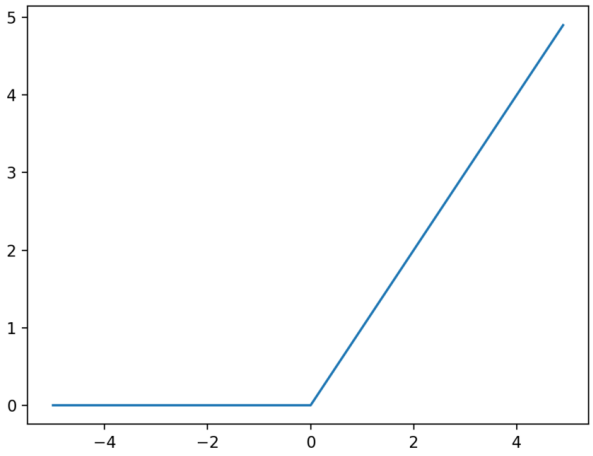

Fig 2.6: ReLU function

is simple, it has been an important achievement in recent years. It has the following advantages:

- Solved the gradient vanishing problem (in the positive range)
- The calculation speed is breakneck, the ReLU function only need to judge whether the input is greater than 0
- The convergence speed is much faster than Sigmoid and Tanh

ReLU also has several issues that require special attention:

- The output of ReLU is not zero-centered
- The Dead ReLU Problem refers to the fact that some neurons may never be activated, causing the corresponding parameters never to be updated. Two reasons cause this situation: (1) Highly unappropriated parameter initialization, which is relatively rare. (2) The learning rate is too high, which leads to too much parameter update during the training process, which unfortunately makes the network enter this state. The solution for these two situations is to use the Xavier initialization method and avoid setting the learning rate too large or using optimizers such as AdaGrad that automatically adjust the learning rate.

Despite these two problems, ReLU is still the most used activation function.

2.2.3 Pooling layer

Compared with the complexity of the convolutional layer, the pooling layer is much simpler. The pooling compresses each sub-matrix of the input tensor. If the layer is 2x2 pooling, then every 2x2 element of the sub-matrix becomes an element. If it is 3x3 pooling, then every 3x3 element of the sub-matrix becomes an element so that the dimensions of the input matrix get smaller.

A pooling standard is needed to turn every nxn element of the input sub-matrix into an element. There are two common pooling standards, MAX or Average. Both of which take the maximum or average value of the corresponding area as the element value after pooling.

The following example uses the maximum pooling method. In addition, 2x2 pooling is used. The stride is 2.

First, the red 2x2 area is pooled. Since the maximum value of this 2x2 area is 6, the value of the corresponding pooling output position is 6. Since the stride is 2, the pooling process moves to the green position for pooling at this time and outputs the maximum value of 8. In the same way, the output values of the yellow and blue areas can be obtained. In the end, the input 4x4 matrix is compressed and becomes a 2x2 matrix after pooling.

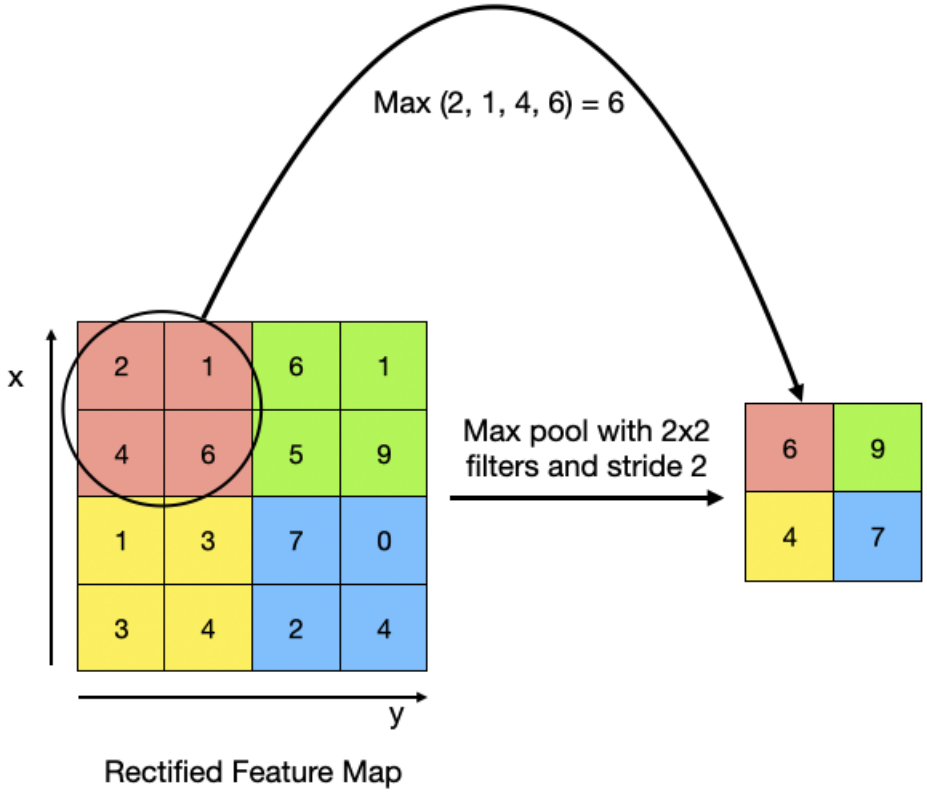

Fig 2.7: Max Pooling

2.2.4 Fully Connected Layer

Fully connected (FC) layers play the role of "classifiers" in the entire convolutional neural network [108]. If operations such as the convolutional layer, pooling layer, and the activation function layer map the original data to the hidden layer feature space, the fully connected layer plays the role of mapping the learned "distributed feature representation" to

the sample label space. In actual use, the fully connected layer can be realized by a convolution operation: the fully connected layer that is fully connected to the previous layer can be transformed into a convolution with a convolution kernel of 1x1, and the fully connected layer that is the convolutional layer in the previous layer can be transformed into the convolution kernel, which is the global convolution of hxw, where h and w are the height and width of the previous convolution result, respectively.

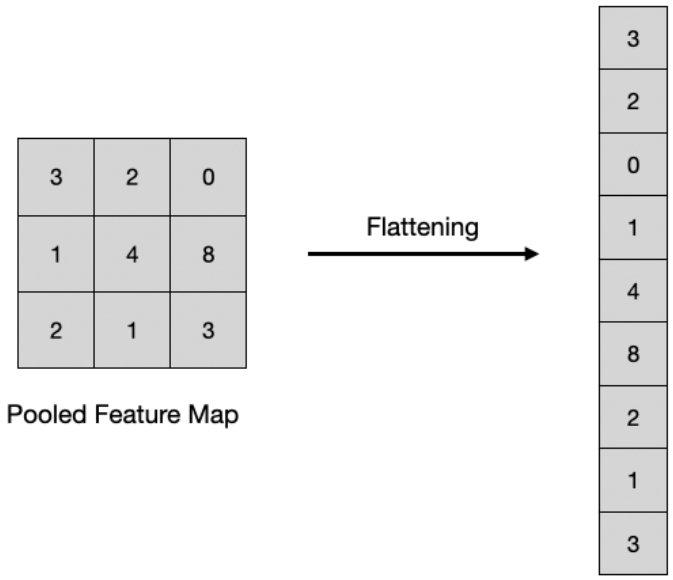

Fig 2.8: Fully Connected Layer

At present, due to the redundancy of fully connected layer parameters (the fully connected layer parameters account for about 80% of the entire network parameters), some recent network models with excellent performance, such as ResNet and GoogLeNet, use global average pooling (GAP) to replace FC to fuse the learned deep features, and then, finally, use softmax and other loss functions as the network objective function to guide the learning process.

2.3 Recurrent Neural Networks, Long Short-Term Memory, Gated Recurrent Units

2.3.1 Recurrent Neural Networks

For the CNN mentioned in the last subsection, the input and output of the training sample are relatively specific. However, there is a kind of problem that CNN is not easy to solve, the training sample input is a continuous sequence, and the length of the sequence is different, such as a time-based sequence with segments of continuous speech and segments of continuous handwritten text, for example. These sequences are relatively long and have different lengths, and it is difficult to split them into independent samples for training through CNN directly.

For this type of problem, RNN [61] is better. RNN assumes that our sample is based on the sequence. For example, for any sequence index number t from sequence index 1 to sequence index T, its corresponding input is $x(t)$ in the corresponding sample sequence. The hidden state of the model at the sequence index number t position $h(t)$ is determined by $x(t)$ and the hidden state at $t-1$ position $(t-1)$ joint decision. At any sequence index number t, the corresponding model prediction output is $o(t)$. By predicting the output $o(t)$ and the real output of the training sequence $y(t)$, and the loss function $L(t)$, CNN-like methods can be used to train the model and then used to predict the output of some positions in the test sequence.

Figure 2.9 below shows how the RNN works.

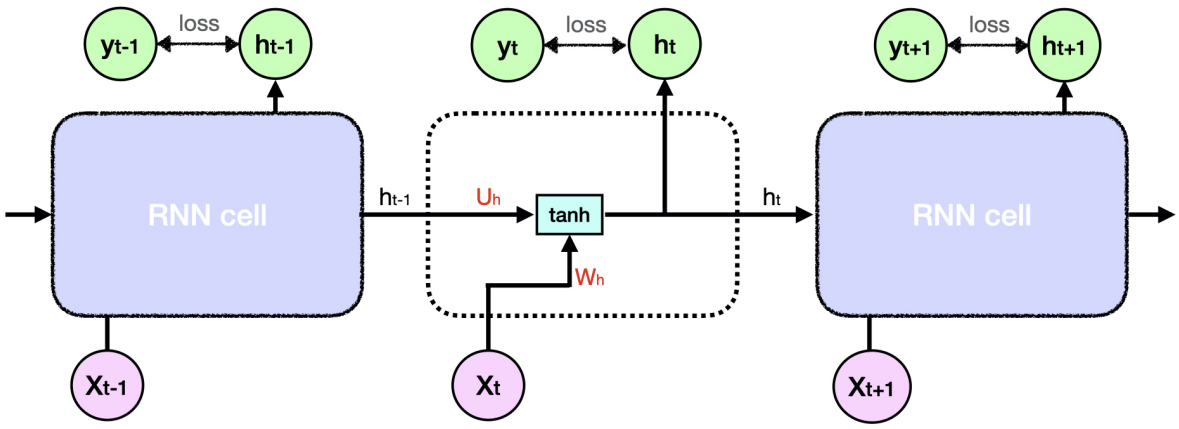

Fig 2.9: RNN model

For any sequence index number t, our hidden state $h(t)$ is obtained by $x(t)$ and $h(t,,1)$:

$$h_t = \sigma\Big(z_t\Big) = \sigma\Big(W_h x_t + U_h h_{t-1} + b_h\Big) \tag{2.5}$$

where σ is active function of RNN, the active function sets $tanh$ generally; h_t is the hidden state at time t; x_t is the input at time t, and h_{t-1} is the hidden state at time $t-1$. W_h and U_h are the weights to be trained, and b_h is the bias.

The output of the model when the sequence index number t is $o(t)$, and the expression is relatively simple:

$$o_t = V h_t + c \tag{2.6}$$

At the end of the sequence index number t, the prediction output is:

$$y'_t = \sigma(o_t) \tag{2.7}$$

Usually, because RNN is a classification model for recognition classes, the above activation function is generally softmax.

Through a loss function $L(t)$, such as the log-likelihood loss function, loss of the model at the current position can be quantified as the distance between $y'_(t)$ and $y_(t)$.

With the basis of the RNN forward propagation algorithm, it is easy to derive the flow of the RNN back propagation algorithm. The idea of the RNN back-propagation algorithm is the same as that of CNN. To acquire the appropriate RNN model parameters U,W,V,b,c, iteration of the gradient descent method has to process. Since back-propagation through time is used, the back-propagation of RNN is sometimes called BPTT (back-propagation through time). Of course, the BPTT and CNN here are also very different because, in BPTT, all the parameters (U, W, V, b, c) are shared at various positions in the sequence, and the same parameters are updated during back-propagation.

Although RNN can solve the training of sequence data beautifully in theory, it also has shortcomings. These two shortcomings cause it to generally not be directly used in NLP fields such as speech recognition, handwritten books, and machine translation. For example,

- Long Term Dependencies: In the field of deep learning (especially RNN), the problem of "long-term dependence" is widespread. The reason for the long-term dependence is that when the nodes of the neural network have been calculated in many stages, the characteristics of the previous relatively long time slice have been covered.

- Gradient Vanishing/Exploding: Gradient disappearance and gradient explosion are two of the critical problems that plague RNN model training. The gradient disappearance and gradient explosion are caused by the cyclic multiplication of the weight matrix of the RNN. Multiple combinations of the same function will lead to extreme nonlinear behavior. The gradient disappearance and gradient explosion mainly exist in RNN, because each time slice in RNN uses the same weight matrix. For a DNN, although it also involves the multiplication of multiple matrices, the problem of gradient disappearance and gradient explosion can be avoided by carefully designing the ratio of weights.

  Gradient truncation can be used to deal with gradient explosion. The gradient cut-off manually reduces the gradient whose gradient value exceeds the threshold θ. Although the gradient truncation will change the direction of the gradient to a certain extent, the direction of the gradient truncation is still the direction in which the loss function decreases.

  Compared with gradient explosion, gradient disappearance cannot be solved simply by a threshold method like gradient truncation because the phenomenon of long-term dependence will also produce a small gradient.

2.3.2 Long Short-Term Memory

The motivation of LSTM [60] is to solve the Long-Term Dependencies problem and the Gradient Vanishing/Exploding problem mentioned above. LSTM can solve the long-term dependence of RNN because LSTM introduces a gate mechanism to control the circulation and loss of features.

Figure 2.10 shows how the LSTM works.

LSTM has a more complex structure than RNN. When the input is time series data, each

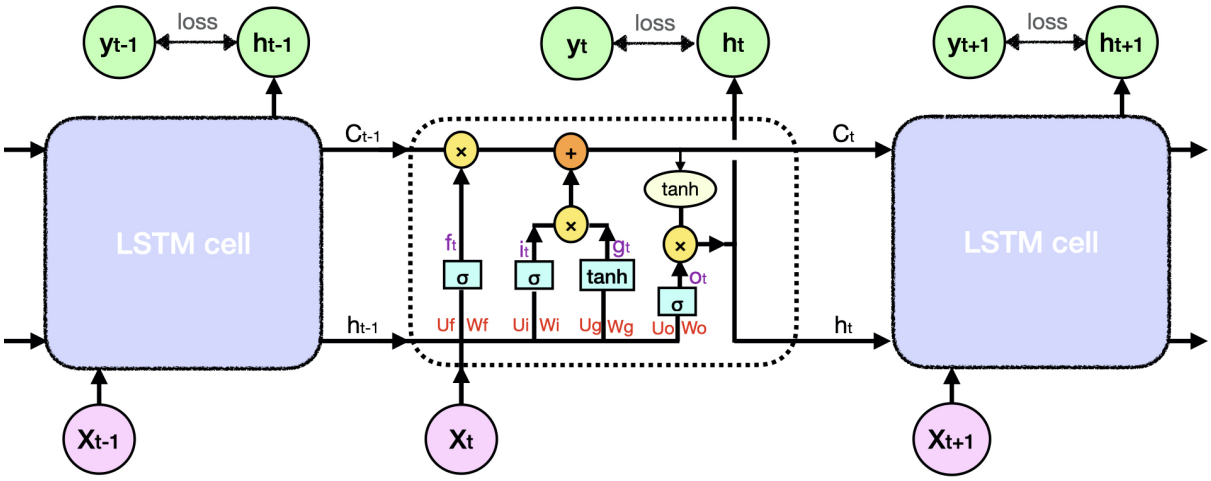

Fig 2.10: LSTM model

cell computes with the following function:

$$\begin{cases} f_t = \sigma(W_f x_t + b_f + U_f h_{t-1} + b'_f) \\ i_t = \sigma(W_i x_t + b_i + U_i h_{t-1} + b'_i) \\ g_t = tanh(W_g x_t + b_g + U_g h_{t-1} + b'_g) \\ o_t = \sigma(W_o x_t + b_o + U_o h_{t-1} + b'_o) \\ c_t = f_t \odot c_{t-1} + i_t \odot g_t \\ h_t = o_t \odot tanh(c_t) \end{cases} \tag{2.8}$$

where h_t is the hidden state at time t; c_t is the cell state at time t; x_t is the input at time t, and h_{t-1} is the hidden state at time $t-1$. f_t, i_t, g_t and o_t are forget gates, input gate 1, input gate 2, and output gates, respectively. W_f, W_i, W_g, W_o and U_f, U_i, U_g, U_o are the weights to be trained, and b_f, b_i, b_g, b_o and b'_f, b'_i, b'_g, b'_o are the bias.

2.3.3 Gated Recurrent Units

The motivation for using GRU [109, 110] is also to solve the Long-Term Dependencies problem and Gradient Vanishing/Exploding problem. However, the LSTM model has lots of weights to calculate. Moreover, GRU has a similar performance to LSTM, and the model does not have many weights. Figure 2.11 shows how the GRU works.

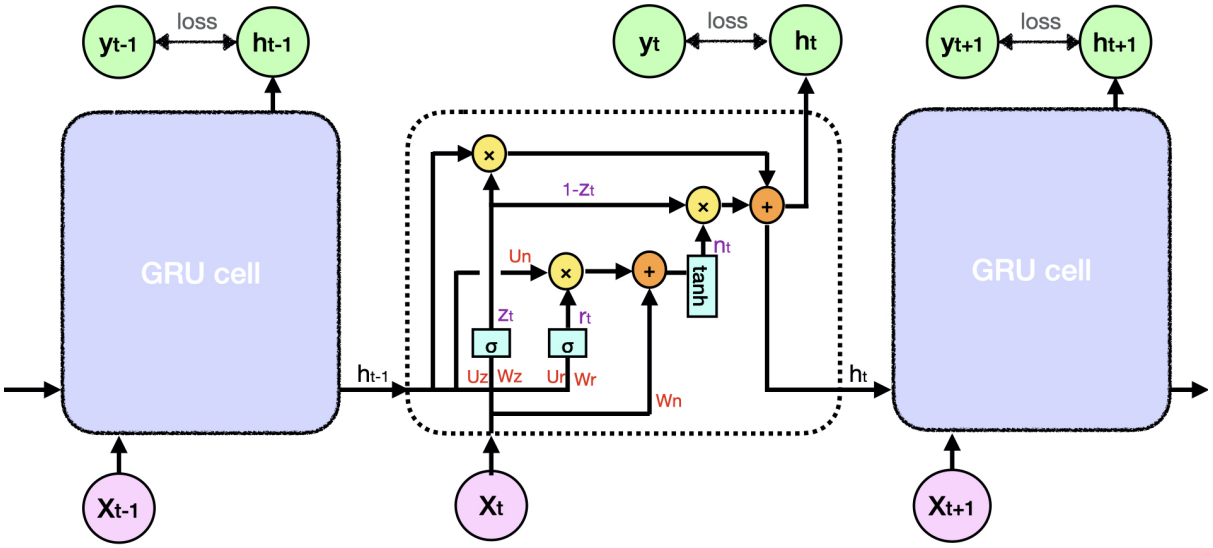

Fig 2.11: GRU model

Each cell of GRU computes with the following function:

$$\begin{cases} z_t = \sigma(W_z x_t + b_z + U_z h_{t-1} + b'_z) \\ r_t = \sigma(W_r x_t + b_r + U_r h_{t-1} + b'_r) \\ n_t = tanh(W_n x_t + b_n + r_t * (U_n h_{t-1} + b'_n)) \\ h_t = (1 - z_t) * n_t + z_t * h_{t-1} \end{cases} \tag{2.9}$$

where h_t is the hidden state at time t; x_t is the input at time t, and h_{t-1} is the hidden state at time $t-1$, and z_t, r_t, n_t are the update, reset and new gates, respectively. W_z, W_r, W_n and U_z, U_r, U_n are the weights to be trained, b_z, b_r, b_n and b'_z, b'_r, b'_n are the bias.

In the LSTM and GRU models, σ is the sigmoid function; $\odot$ is the Hadamard product. The Hadamard product of an m × n matrix A and an m × n matrix B is marked as A $\odot$ B. For example, A 3 × 2 matrix A $\odot$ a 3 × 2 matrix B with the flowing function.

The structure of GRU input and output is like an ordinary RNN, and the interior idea is similar to LSTM. Compared with LSTM, there is no "gating" mechanism inside the GRU, which has fewer parameters than LSTM, but it can also achieve functions equivalent to LSTM. Considering the computing power and time cost of the hardware, GRU can also be used in many cases to solve the time series forecasting problem.

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{pmatrix} \odot \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{pmatrix} = \begin{pmatrix} a_{11}b_{11} & a_{12}b_{12} \\ a_{21}b_{21} & a_{22}b_{22} \\ a_{31}b_{31} & a_{32}b_{32} \end{pmatrix} \tag{2.10}$$

2.4 Generative Adversarial Networks (GANs)

Generative Adversarial Networks (GANs) [14] is an unsupervised machine learning framework which is used in the theoretic of game theory. This framework trains two differentiable networks: Generator G to capture the data distribution and Discriminator D to discriminate whether the data is real data or generated data. Although GANs was originally proposed for unsupervised learning, it has also been shown to be useful for semi-supervised learning, supervised learning, and reinforcement learning. Yann LeCun, an outstanding representative in the field of deep learning, wrote on Quora: "Generative Adversarial Networks (GANs) and related changes are what I think it is the most interesting idea in the field of machine learning in the past decade."

The training of GANs can be analogized to the process of playing chess. If a chess player wants to improve his chess playing skill, then his opponent who plays chess has to be more skillful than him. Then in the process of playing chess the player needs to learn what he does not do well and what his opponent does well, so that he can think how to beat his opponent. This concept can be incorporated into building better models. So in simple terms, to acquire a powerful skill in playing chess (like Generator), the model need a stronger opponent (like Discriminator).

Another analogy similar to the relationship between the Discriminator and the Generation in GANs is the relationship between the forger (Generator) and the investigator (Discriminator). The task of the forger is to imitate the paintings of famous artists. If the forged work can surpass the original work, then the forger can sell the work for a lot of money. On the other hand, the task of art investigators is to catch these forgers. The competition between the counterfeiters and the investigators continued, eventually giving birth to high skilled counterfeiters and investigators.

2.4.1 How GANs work

The Generator network and the Discriminator network are shown in Figure 2.12.

The Generator network uses random inputs and tries to output "realistic-looking" data samples. In the image below, the Generator $G(z)$ gets the input z from $p_z(z)$, where z is a sample from the probability distribution $p(z)$. The Generator generates a piece of data

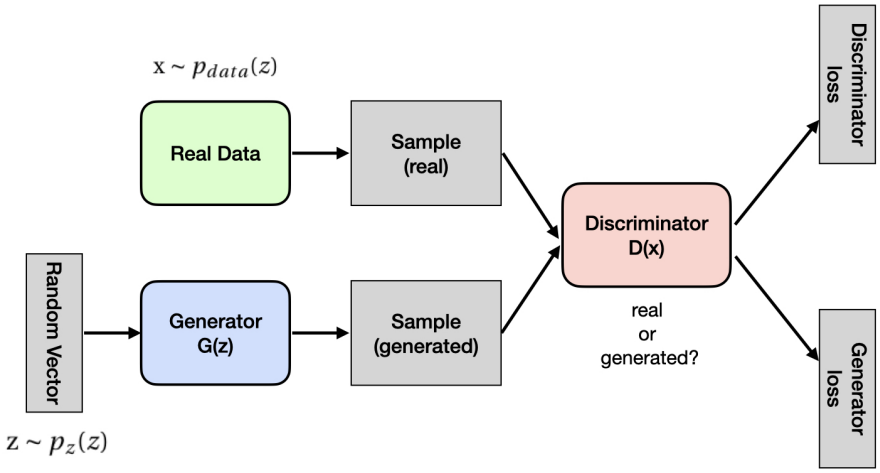

Fig 2.12: Generative Adversarial Networks

and passes it to the Discriminator network $D(x)$. The task of the Discriminator network is to receive real data or to generate data, and to try and predict whether the input is real or generated. This task requires an input x from $p_{data}(x)$, where $p_{data}(x)$ is the true data distribution. $D(x)$ then uses the Sigmoid function to solve the binary classification problem and outputs a value from 0 to 1.

The loss function of GANs is the Min-Max loss function:

$$\min_G \max_D V\Big(D,G\Big) = E_{x\sim p_{data}(x)}\Big[\log D(x)\Big] + E_{z\sim p_z(z)}\Big[\log D\Big(1-G\Big(z\Big)\Big)\Big] \tag{2.11}$$

The Generator network and Discriminator network are not trained simultaneously. The training strategy is to use **separate alternating iterative training**.

To use this training for the Discriminator: Assume that the Generator network model is already available (however, it may not be the best Generator network), then given random inputs, the output will receive fake sample data, as well as true sample sets. Next, the labels of true and false samples are artificially defined because the output of the true sample data should be as close to 1 as possible, and the false sample data to 0. Obviously, to Discriminator all the labels of the true sample data are defaulted to 1, and all the labels of the

fake sample data are defaulted to 0. In this way, in terms of the Discriminator network, the problem becomes a simple supervised two-classification problem, which is directly passed to the network model for training. Figure 2.13 shows this process of training the Discriminator network. Assuming that the training of Discriminator network is complete, and then it is the turn for the Generator network training.

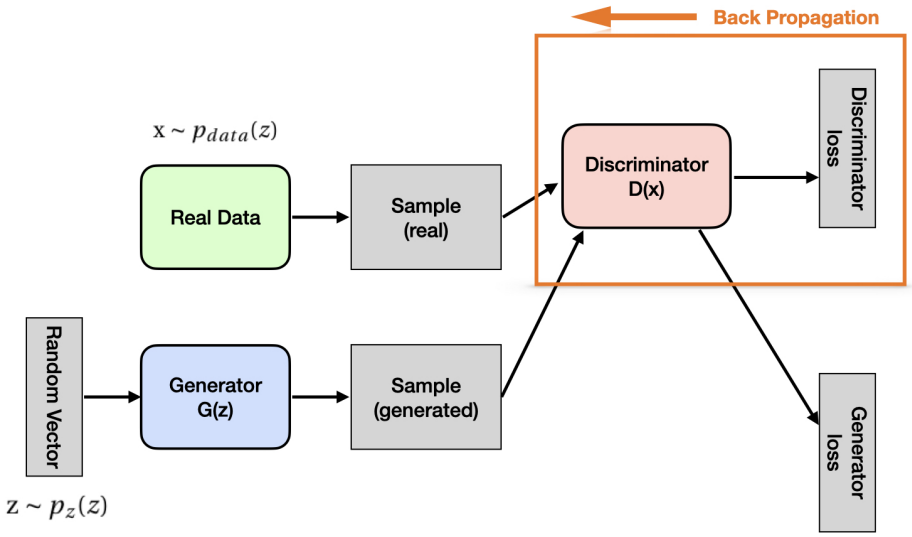

Fig 2.13: Generative Adversarial Networks

The loss function of Discriminator is shown below:

$$\max_D V\Big(D,G\Big) = E_{x\sim p_{data}(x)}\Big[\log D(x)\Big] + E_{z\sim p_z(z)}\Big[\log D\Big(1-G\Big(z\Big)\Big)\Big] \tag{2.12}$$

To use this training for the Generator: generate samples that are as realistic as possible. Judging the authenticity of the generated samples requires the Discriminator network, so when training the generated network, the Discriminator network must be used together with the Generator to achieve the purpose of training. If only use the Generator network to train, there will be no loss to do the back propagation. But if the Discriminator network that has just been trained is connected to the back of the Generator network, there will be losses for training. Therefore, the training of the Generator network is training of the Generator-Discriminator network concatenated, as shown in Figure 2.14. Currently, the key point is

that **the labels of these fake samples should be set to 1**, which means that these fake samples are real samples in the Generator network training. Only in this way the Generator can serve the purpose of confusing the discriminator, and can also make the generated false samples gradually approach the real samples. To train the Generator network, sample data (only a fake sample set, and not a real sample set), with the corresponding label (all 1) is used. When training this concatenated network, **don't discern the changes in the parameters of the Discriminator network**, which means that the parameters should be prevented from being updated, but the loss should be passed all the way to the Generator network. This completes the training of the Generator network.

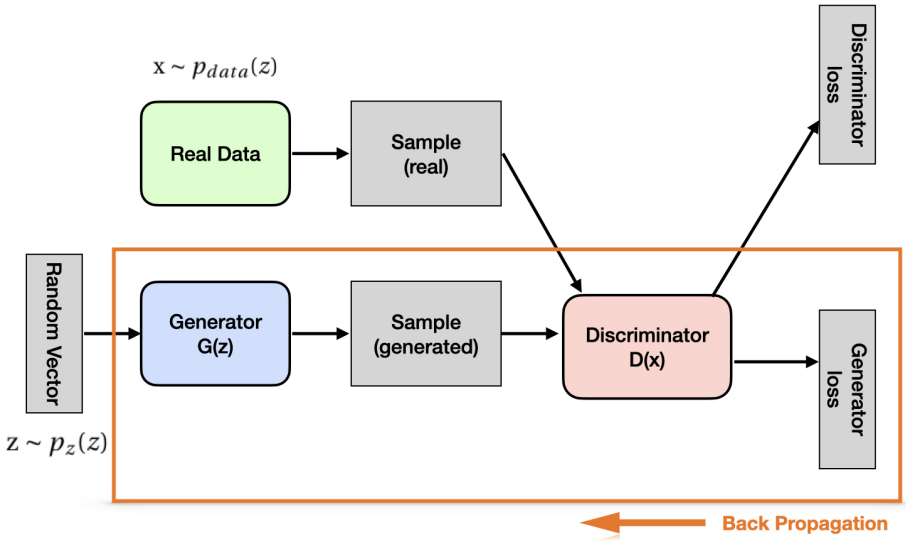

Fig 2.14: Generative Adversarial Networks

The loss function of Generator is shown below:

$$\min_G V\left(D,G\right) = E_{z\sim p_z(z)}\left[\log D\left(1-G\left(z\right)\right)\right] \tag{2.13}$$

After completing the Generator network training, new fake samples can be generated from the previous random input z based on the current new Generator network, and the fake samples after training are more like real data. Then there is a new set of true and false samples (in fact, a new set of false samples), so that the above process can be repeated. This

process is called separate alternating iterative training. A number of iterations can be defined and stopped after a certain number of alternate iterations. One of the most important points is the transformation from false samples to real samples in the training process.

In summary, the iteration training of GANs includes two processes completed in sequence:

- Training the Discriminator, fixing the Generator (fixing means not training, the network only propagates forward, not back propagation)
- Training the Generator and fixing the Discriminator.

There are several steps when using the GANs framework to generate "realistic-looking" data:

- Step 1: Define the problem. The problem is like generate fake images or text. The problem should be fully defined and collect data.
- Step 2: Define the GANs architecture. According to the problem, different differentiable networks should be set in Generator and Discriminator.
- Step 3: Use real data to train the discriminator for N epochs. Train the discriminator to correctly predict that the real data is true. Here N can be set to any natural number between 1 and infinity.
- Step 4: Use the generator to generate fake input data to train the discriminator. Train the discriminator to correctly predict false data as false.
- Step 5: Train the generator with the access of the discriminator. When the discriminator is trained, its predicted value is used as a marker to train the generator. Train the generator to confuse the discriminator.
- Step 6: Repeat steps 3 to 5 for multiple epochs
- Step 7: Manually check whether the fake data is reasonable. Stop training if it seems appropriate, otherwise go back to step 3. This is a manual task, and manual evaluation of data is the best way to check the degree of counterfeiting. When this step is over, the researcher can evaluate whether the GANs are performing well.

Algorithm 1 Mini-batch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyper-parameter.

for number of training iterations **do**

for k steps **do**

- Sample mini-batch of m noise samples $\{z^{(1)},..., z^{(m)}\}$ from noise prior $p_g(z)$.
- Sample mini-batch of m examples $\{x^{(1)},..., x^{(m)}\}$ from data generating distribution $p_{data}(x)$.
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(x^{(i)}\right) + log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right)\right] \tag{2.14}$$

end for

- Sample mini-batch of m noise samples $\{z^{(1)},..., z^{(m)}\}$ from noise prior $p_g(z)$.
- Update the Generator by ascending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right) \tag{2.15}$$

end for

The gradient-based updates can use any standard gradient-based learning rule.

Compared with other generative models, GANs has the following advantage:

- GANs can generate samples faster than completely obvious belief networks (NADE, PixelRNN, WaveNet, etc.), because it does not need to generate different data in the sampling sequence.
- GANs do not require Monte Carlo estimation to train the network. People often complain that GANs are unstable and difficult to train, but they are much simpler than training Boltzmann machines that rely on Monte Carlo estimation and logarithmic partition functions. The Monte Carlo method does not work well in high-dimensional spaces. Boltzmann machines have never been extended to tasks like Image-Net. GANs can at least learn to draw some fake images after training on Image-Net.
- Compared to variational autoencoders, GANs do not introduce any deterministic bias, and variational methods introduce deterministic bias, because they optimize the lower bound of the log likelihood, rather than the likelihood itself, which seems to cause the instances generated by VAEs, which are more obscure than GANs.
- Compared with non-linear ICA (NICE, Real NVE, etc.), GANs do not require the latent variables input by the generator to have any specific dimensions or require the generator to be reversible.
- Compared with Boltzmann machines and GSNs, the process of generating instances of GANs only requires the model to run once, rather than iterating many times in the form of Markov chains.

2.5 Genetic Algorithm

The Genetic Algorithm (GA) [111] originated from the computer simulation study of biological systems. It is a stochastic global search and optimization method developed by imitating the biological evolution mechanism of nature, drawing lessons from Darwin's theory of evolution and Mendel's theory of genetics. Its essence is an efficient, parallel and global search method, which can automatically acquire and accumulate knowledge about

the search space during the search process, and adaptively control the search process to find the best solution.

Relevant terms are given below:

- Genotype: the internal manifestation of trait chromosomes.
- Phenotype: the external performance of the trait determined by the chromosome, or the external performance of the individual formed according to the genotype.
- Evolution: The population gradually adapts to the living environment, and the quality is continuously improved. The evolution of organisms is carried out in the form of populations.
- Fitness: a measure of the degree of adaptation of a species to the living environment.
- Selection: Select several individuals from the population with a certain probability. Generally, the selection process is a process of survival of the fittest based on fitness.
- Reproduction: When a cell divides, the genetic material DNA is transferred to the newly produced cell through replication, and the new cell inherits the genes of the old cell.
- Crossover: DNA is cut at a certain position of two chromosomes, and the front and back strings are cross-combined to form two new chromosomes, which is called genetic recombination or hybridization.
- Mutation: It is possible (with a very small probability) that some replication errors will occur during replication, and mutation will produce new chromosomes, showing new traits.
- Individual: this term refers to the entity with characteristics of the chromosome.
- Population: a collection of individuals, the number of individuals in the collection is called the size of the population.

Each chromosome in the genetic algorithm corresponds to a solution of the genetic algorithm. Generally, we use the fitness function to measure the pros and cons of this solution. So, the fitness from a genome to its solution forms a mapping. The process of a genetic algorithm is a process of finding the optimal solution in a multivariate function.

The realization process of genetic algorithm is like the evolution process in nature. First, look for a scheme to "digitally" encode the potential solution of the problem (Establish the mapping relationship between phenotype and genotype). Then initialize a population with random numbers, and the individuals in the population are these digital codes. Next, after an appropriate decoding process, the fitness function is used to evaluate the fitness of each individual gene. Use the selection function to select the best according to a certain regulation. Let individual genes mutate.

Then produce offspring. The biggest advantage of using a genetic algorithm is that it is easy to "find" the optimal solution; that is, simply "negate" some individuals who are not performing well.

The general steps of the genetic algorithm are summarized as follows:

Start the cycle until a satisfactory solution is found.

1) Assess the fitness of the individual corresponding to each chromosome.

2) Following the principle that the higher the fitness, the greater the probability of selection, select two individuals from the population as the parents.

3) Extract the chromosomes of both parents and cross over to produce offspring.

4) Make mutations to the chromosomes of the offspring.

5) Repeat steps 2, 3 and 4 until a new population is produced.

End the loop.

2.6 Comparative Algorithm

Linear Models

Traditional linear models are explained in this section. The Autoregressive Average (AR) model, the Moving Average (MA) model, the Autoregressive Moving Average (ARMA) model

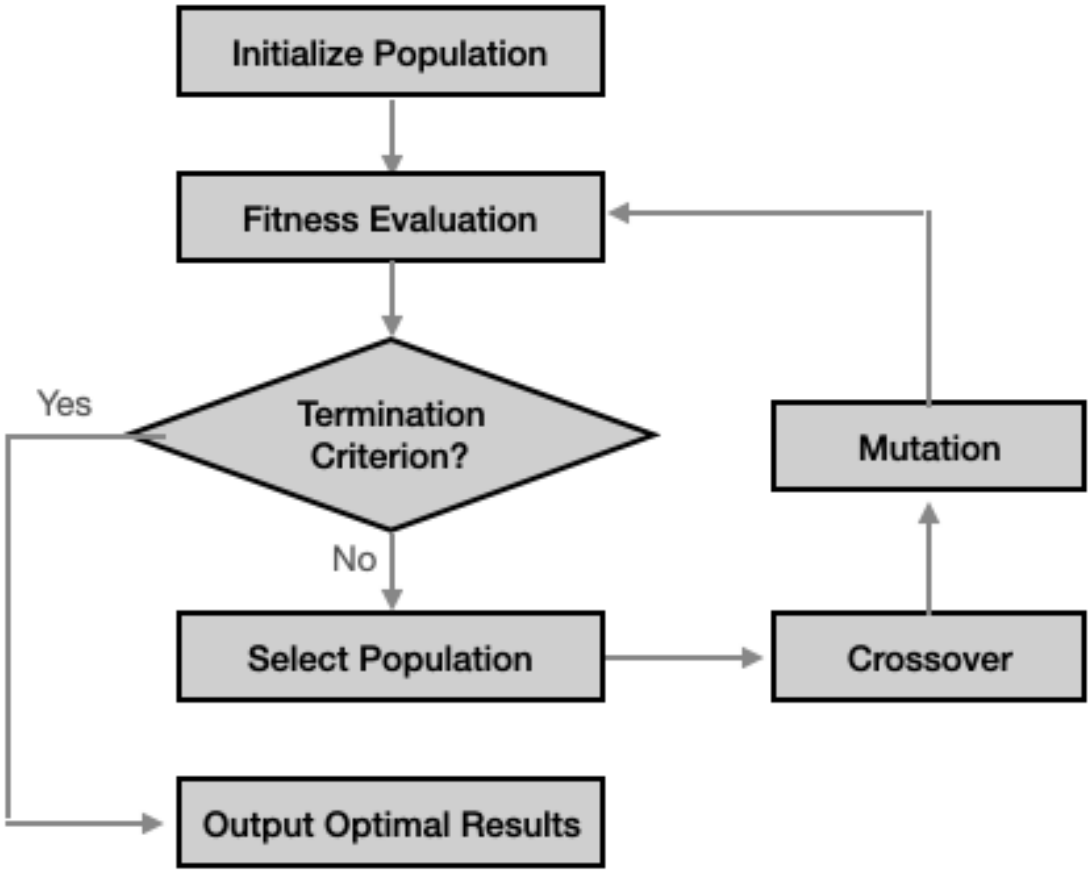

Fig 2.15: Genetic Algorithm

and the Autoregressive Integrated Moving Average model (ARIMA) are described and then, the algorithm to determine the model parameters are described.

2.6.1 Autoregressive Average (AR) model

The AR model, which is referred to as AR(p), represents the stock price X_t by the past stock price X_{t-1} and the error term ϵ_t.

$$X_t = c + \alpha_0 + \sum_{i=1}^{p} \alpha_i X_{t-i} + \epsilon_t \tag{2.16}$$

where t and c and α_i denote the data and the model parameter, respectively. The error term u_t depends on the white noise of average 0 and the volatility σ^2.

2.6.2 Moving Average (MA) model

The MA model, which is referred to as MA(q), represents the stock price X_t by the error term ϵ_t:

$$X_t = c + \epsilon_0 + \sum_{i=1}^{q} \beta_j \epsilon_{t-j} \tag{2.17}$$

where β_j denotes the model parameter. The error term u_t depends on the white noise of average 0 and the volatility σ^2.

2.6.3 Autoregressive Moving Average (ARMA) model

The ARMA model, which is referred to as ARMA(p,q), is the combination model of the AR and MA models:

$$X_t = c + \epsilon_0 + \sum_{i=1}^{p} \alpha_i X_{t-i} + \sum_{i=1}^{q} \beta_j \epsilon_{t-j} \tag{2.18}$$

2.6.4 Autoregressive Integrated Moving Average (ARIMA) model

The ARIMA model, which is referred to as ARIMA(p,d,q), is the combination model of the AR and MA models:

$$X_t = X_{t-d} + \epsilon_0 + \sum_{i=1}^{p} \alpha_i X_{t-i} + \sum_{i=1}^{q} \beta_j \epsilon_{t-j} \tag{2.19}$$

2.6.5 Model Parameter Estimation

The model parameters are determined to minimize the Akaike Information Criteria (AIC):

$$AIC = 2\ln\sigma + 2\frac{p+q+d}{T} \tag{2.20}$$

Chapter 3

Stock Price Prediction

3.1 Introduction

In European and American capital markets, algorithmic trading has been deeply involved in quantitative investment strategies and has become one of the most important financial technology applications. In the 1990s, algorithmic trading began to be applied to stock brokerage business in the United States. With the continuous improvement of algorithmic trading platforms and technological advancement, in recent years, the US stock market has applied the algorithmic trading execution business on a large scale. A study in 2016 showed that over 80% of trading in the FOREX market was performed by trading algorithms rather than humans. Since the global financial crisis in 2008, quantitative investment has received widespread attention, and algorithmic trading has gradually merged with quantitative investment.

In the area of Quantitative Finance, the Autoregressive Integrated Moving Average model (ARIMA), one of the methods of time series forecast analysis, was widely used before the neural networks. The ARIMA model has three parameters(p, d, q), which represent the order of the autoregressive model, the degree of differencing and the order of the moving-average model, respectively. Although the ARIMA model is simple enough to need only endogenous variables and no need to use other exogenous variables, it is a linear model, but it has two fatal demerits:

- The time series data is required to be stationary, or it must be stable after being differencing.

- Essentially, time-series data can only capture linear relationships in the time series dataset, not nonlinear relationships.

Long Short-Term Memory (LSTM) [60] is one of the Recurrent Neural Networks (RNN) [61] architecture. Unlike ARIMA, it can mine the nonlinear information from time series data like stock market data. At first, LSTM was used for Natural Language Processing (NLP). Now it can be used with speech recognition tasks and time-series tasks for capturing deep patterns from these datasets.

In this research, the architecture is based on the time series neural networks which are RNN, LSTM, and Gate Recurrent Unit (GRU) [109] and another state-of-art network called Generative Adversarial Networks (GANs). Goodfellow has presented generative Adversarial Networks (GANs) in 2014 [14]. GANs are classified into one of the unsupervised machine learning algorithms. GANs have outstanding performance in the field of image generation. Currently, GANs can also generate text and speech data by changing the inside architecture of GANs.

GANs have two networks: a Generator and a Discriminator. The alterability of this structure makes GANs very flexible. LSTM is chosen in this research as the Generator to capture the inside pattern from time series stock data and generate a future trend, and MLP, RNN, LSTM, and GRU as the Discriminator to distinguish the real future data and the data generated by Generator. In this work, the Discriminator is treated as a huge loss function that consists of another neural network. These hybrid sequential GANs are used to find which GANs model can best perform on the three training strategies.

Two training strategies using the successive-GANs:

- rolling window training
- segmented data training

In this study, the S&P 500 dataset is used for numerical experiments.

3.2 Methodology

In the proposed algorithm, an LSTM model is set without loss function into the Generator, and time-series neural networks (RNN, LSTM, and GRU), MLP into the Discriminator

separately [112]. The Discriminator is regarded as a huge loss function which consists of a time series network and MLP. In addition, a regular loss function must be set (Least squares loss function) to train the Discriminator. Figure 3.1 illustrates the structure of the proposed algorithm.

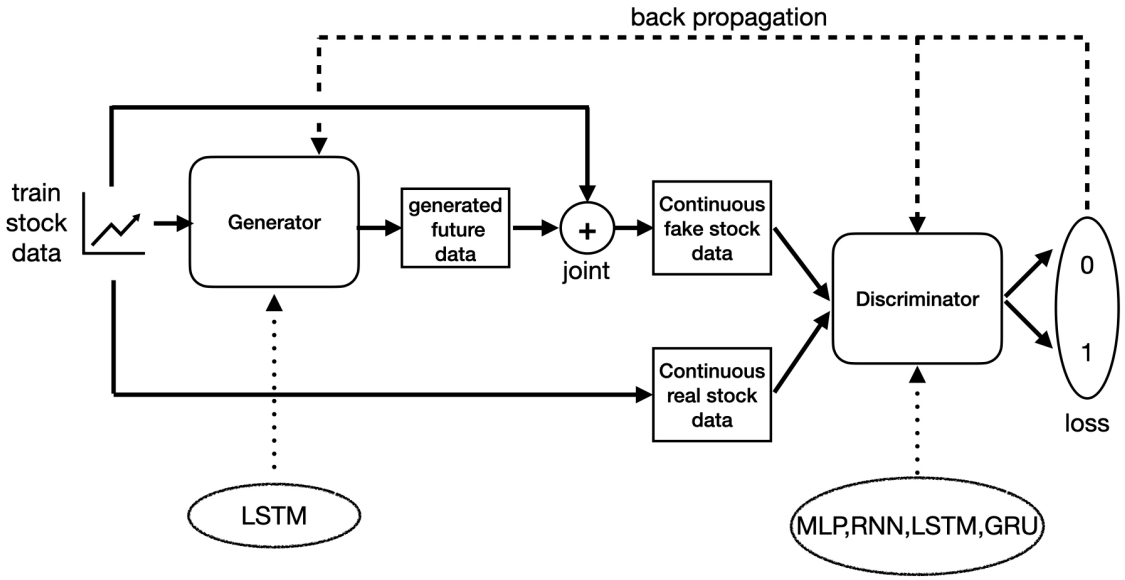

Fig 3.1: Proposed Algorithm

Let (X, Y) be a continuous sample from the stock dataset. For example, X is 5 consecutive days stock data with 6 features ('open', 'close', 'high', 'low', 'volume' and 'adj close') and Y is 2 consecutive days stock data after X with 6 features. Let $G(X)$ be the generated data by the Generator.

3.2.1 Training Discriminator

The input of the Discriminator is $(X,\ Y)$ and $(X,\ G(X))$, and the output is a scalar in the range between 0 and 1. The value 1 and 0 means the input is real or fake, respectively. The Discriminator tries to classify the input $(X,\ Y)$ into class 1 and the input $(X,\ G(X))$ into class 0. Therefore, the Discriminator is more like a binary classification network. When the Discriminator is training, the weights of the Generator have no change, which means the Generator is not training in the meanwhile.

The loss function for training the Discriminator is:

$$L^D = L_{loss}(D(X,Y),1) + L_{loss}(D(X,G(X)),0) \tag{3.1}$$

$$L_{loss} = \sum_{i=1}^{n} (y_{true} - y_{predicted})^2 \tag{3.2}$$

Minimizing this loss function means that the ability of the Discriminator in discriminating the "real" data and "fake" data has been improved.

3.2.2 Training Generator

The input of the Generator is a different "real" continuous stock data sample X, and the output is $G(X)$ generated by Generator. While fixing the weights of Discriminator, the adversarial loss should be minimized as defined below:

$$L^G = L_{loss}(D(X,G(X)),1) \tag{3.3}$$

Minimizing this loss function means that the ability of Generator in cheating the Discriminator has been improved. This means the stock data generated by the Generator is more like the "real" future data.

3.2.3 Process

The algorithmic process is summarized as follows:

1. The weights of the Generator are fixed, and the Discriminator is trained to minimize the loss function to update the weights of the Discriminator.

2. The weights of Discriminator are fixed, and the Generator is trained to minimize the loss function to update the weights of the Generator.

The flow chart of the proposed algorithm is shown in Algorithm 2.

3.3 Experiment

3.3.1 Experimental Dataset

Since not all companies have enough training data in the S&P 500, 50 companies from S&P 500 stock for training and testing the model in this research were selected. The average value

Algorithm 2 successive-GANs

Input Data:
continuous stock data M: (X, Y)
Generator:

- input: X
- output: $G(X)$
- Generator learning rate: l_G

Discriminator:

- input: (X, Y) and $(X, G(X))$
- output: a scalar value between 0 and 1
- Discriminator's learning rate: l_D

for number of training iterations EPOCH **do**
Get one data sample from M, Generator fixed, update the Discriminator weights W_D:

$$W_D = W_D - l_D(\frac{\partial L^D(X,Y)}{\partial W_D} + \frac{\partial L^D(X,G(X))}{\partial W_D})$$

Get another data sample from M, Discriminator fixed, update the Generator weights G_D:

$$W_G = W_G - l_G \frac{\partial L^G(X,G(X))}{\partial W_G}$$

end for

of 50 companies for loss and accuracy was used. The training data are from 2010/01/01 to 2019/11/30 and the test data from 2019/12/01 to 2019/12/31. The dataset contains 6 features ('open', 'close', 'high', 'low', 'volume' and 'adj close'). The dataset is from Yahoo Finance. An example of training data is shown in Figure 3.2.

AAPL

Date	High	Low	Open	Close	Volume	Adj Close
2009-12-31	30.47857093811040	30.079999923706100	30.4471435546875	30.104286193847700	88102700.0	26.131752014160200
2010-01-04	30.64285659790040	30.34000015258790	30.489999771118200	30.572856903076200	123432400.0	26.538482666015600
2010-01-05	30.79857063293460	30.464284896850600	30.657142639160200	30.625713348388700	150476200.0	26.584365844726600
2010-01-06	30.747142791748000	30.10714340209960	30.625713348388700	30.13857078552250	138040000.0	26.161508560180700
2010-01-07	30.285715103149400	29.864286422729500	30.25	30.082857131958000	119282800.0	26.11314582824710
2010-01-08	30.285715103149400	29.86571502685550	30.042856216430700	30.28285789489750	111902700.0	26.286752700805700
2010-01-11	30.428571701049800	29.77857208251950	30.399999618530300	30.015714645385700	115557400.0	26.05486488342290
2010-01-12	29.96714210510250	29.488571166992200	29.88428497314450	29.674285888671900	148614900.0	25.758487701416000
2010-01-13	30.132856369018600	29.157142639160200	29.695714950561500	30.092857360839800	151473000.0	26.121826171875
2010-01-14	30.06571388244630	29.860000610351600	30.015714645385700	29.91857147216800	108223500.0	25.97054672241210
2010-01-15	30.22857093811040	29.40999984741210	30.132856369018600	29.41857147216800	148516900.0	25.53652000427250
2010-01-19	30.74142837524410	29.605714797973600	29.761428833007800	30.719999313354500	182501900.0	26.66621208190920
2010-01-20	30.792856216430700	29.928571701049800	30.70142936706540	30.247142791748000	153038200.0	26.255752563476600

Fig 3.2: AAPL stock data

3.3.2 Training Strategy

Two training strategies use the hybrid sequential GANs:

3.3.2.1 rolling window training Figure 3.3 shows how the rolling window strategy train the hybrid sequential GANs. As an example, numbers of input and output data are given as 5 and 1, respectively. In rolling window training, the stock data on 5 consecutive days are taken as the input data and the stock data on the next day of 5 consecutive days are taken as output data. Therefore, in the first set, the stock data on the 1st to 5th days and the stock data on the 6th day are input and output data, respectively. In the second set, the stock data on the 2nd to 6th days and the stock data on the 7th day are input and output data, respectively.

3.3.2.2 segmented data training Figure 3.4 shows how the segmented data strategy train the hybrid sequential GANs. In this case, the first set contains the stock data on the 1st to

5th days as input and the stock data on the 6th day as output. The second set contains the stock data on the 6th to the 10th days as input and the stock data on the 11th day as output.

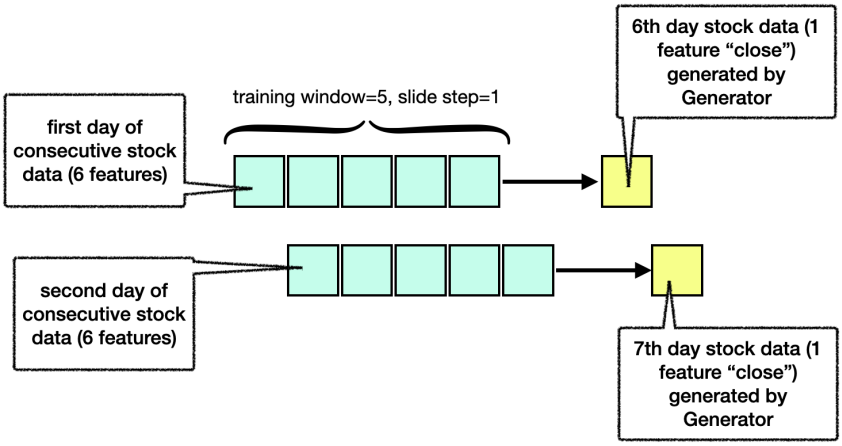

Fig 3.3: rolling window training

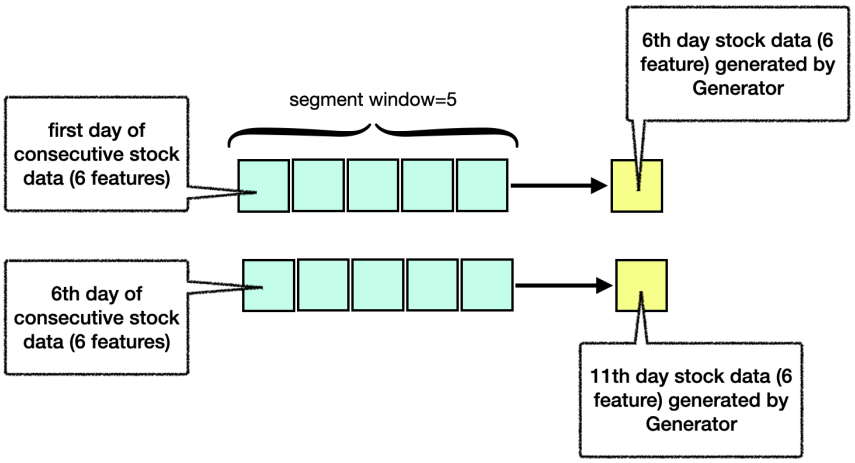

Fig 3.4: segmented data training

Evaluation

When we use the rolling window training and segmented data training, the RMSE and MAE are used for evaluating the model.

Here is the RMSE function:

$$RMSE = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(y_i - y_i{}')^2} \tag{3.4}$$

Here is the MAE function:

$$MAE = \frac{1}{m}\sum_{i=1}^{m}\left|y_i - y_i{}'\right| \tag{3.5}$$

y_i is the real data of stock dataset, y_i' is the generated stock data by sequential GANs, m is the number of data.

Experiment environment

In this work, all codes are written by PyTorch [113] [114], which is an open-source machine learning library. The CPU and GPU that were used are Intel Xeon(R) CPU 3.5GHz, GeForce GTX TITAN X, Intel i7-8700 CPU 3.20GHz, and GeForce GTX 1080.

3.3.3 Experimental Results

Because there are some companies in S&P 500 stocks that do not have 10 years' worth of data, 50 companies were chosen for this research from the S&P 500 stocks for training the model and testing.

For convenience, in the following section D is designated as the Discriminator and G as the Generator. The hybrid sequential GANs include four forms: G-LSTM+D-MLP, G-LSTM+D-RNN, G-LSTM+D-LSTM,and G-LSTM+D-GRU. The words following to G- and D- represent the algorithm used for the Generator and Discriminator, respectively. For example, G-LSTM+D-MLP represents the GANs with LSTM as the Generator and MLP as Discriminator. The hybrid sequential GANs with ARIMA, were compared to the simple LSTM and the simple GANs using the two training strategy which mentioned in the last section.

The following example shows how training loss changed when using the simple LSTM (Figure 3.5) and G-LSTM+D-LSTM (Figure 3.6) model. The training data uses "AAPL"(APPLE) stock for 10 years.

The hyper-parameters used in this example are as follows:

- EPOCH: 100

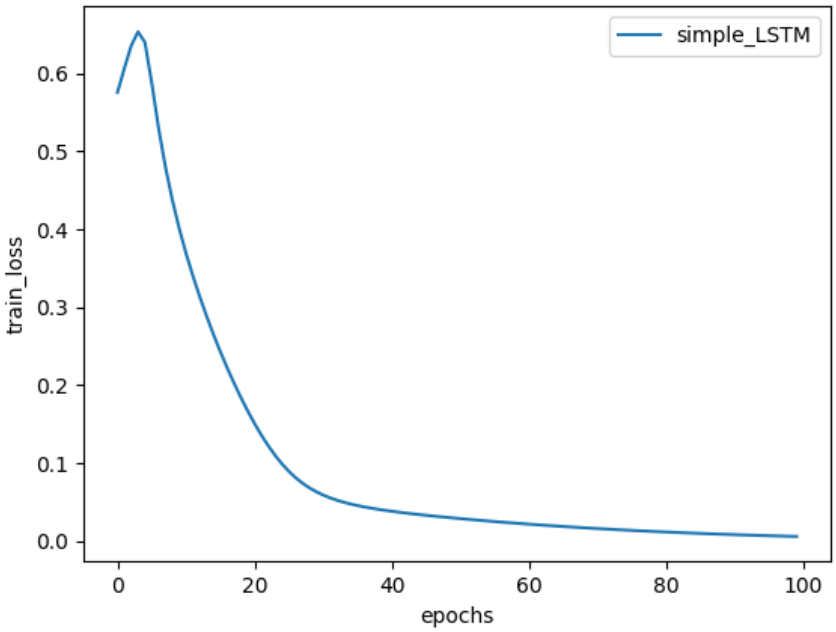

Fig 3.5: simple LSTM train loss

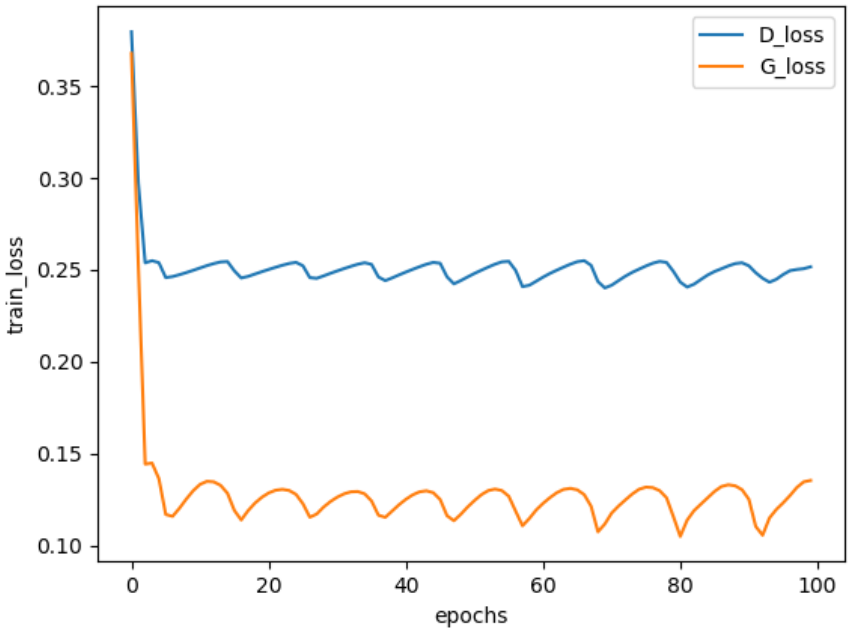

Fig 3.6: G-LSTM+D-LSTM train loss

- BATCH_SIZE: 10
- Learning Rate: 0.0001

When using the simple LSTM for training, the model can be used directly to predict the future stock data. When using the hybrid successive GANs proposed in this research, the Generator is used to predict the future stock data and the Discriminator is abandoned.

Figures 3.7, 3.8, 3.9, and 3.10 show four examples for the prediction comparison by using different models on the test data. The stocks used are Apple, Adobe, AMD, and Google company stock data for 1 month.

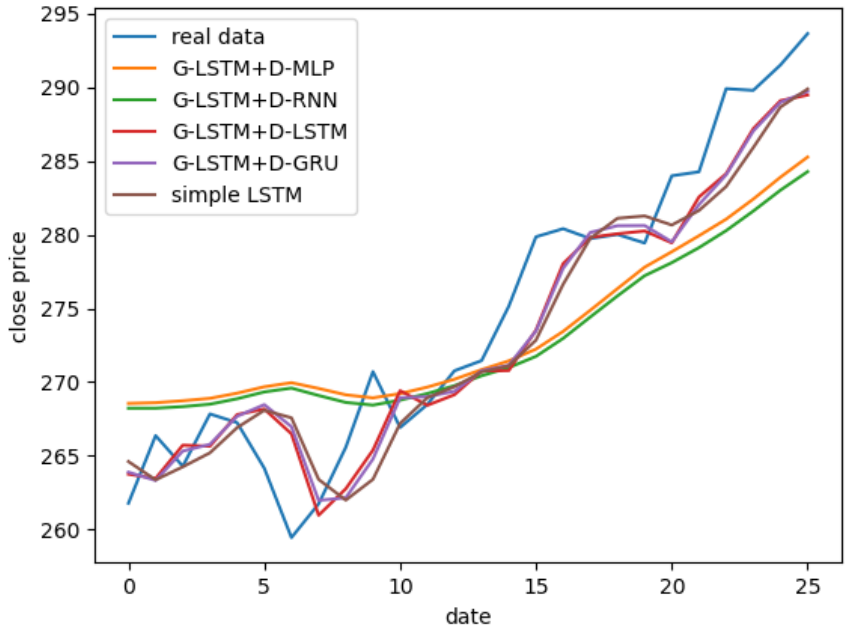

Fig 3.7: Prediction comparison using different models on test data (Apple)

In TABLE 3.1 and TABLE 3.2 different models were compared (ARIMA, GANs, LSTM, hybrid successive GANs) with three different strategies, which were mentioned the section above. The lower the evaluation values of RMSE and MAE the better the performance of model is.

Strategy 1: the rolling window strategy on different model comparisons is shown in TABLE 3.1. The result shows that the G-LSTM+D-LSTM has the best performance. Although

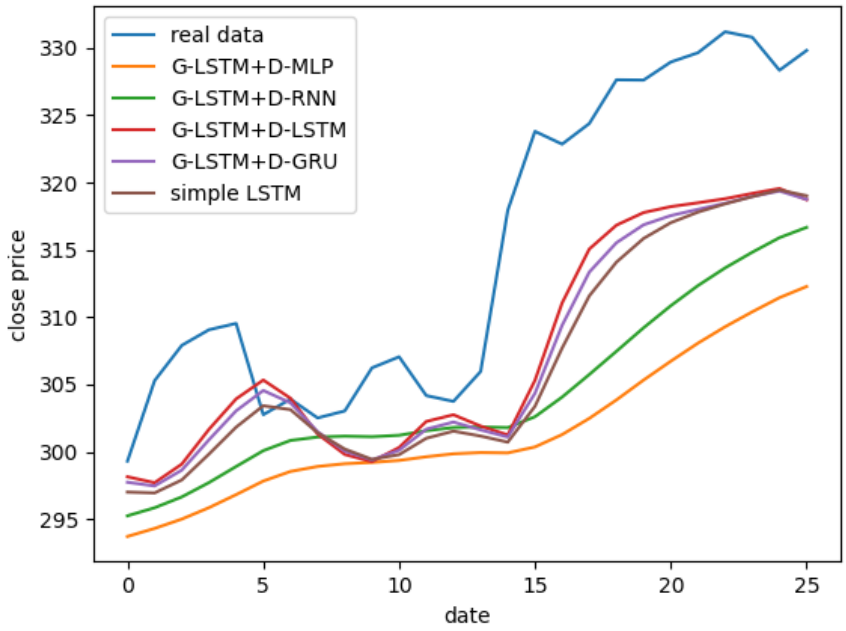

Fig 3.8: Prediction comparison using different models on test data (Adobe)

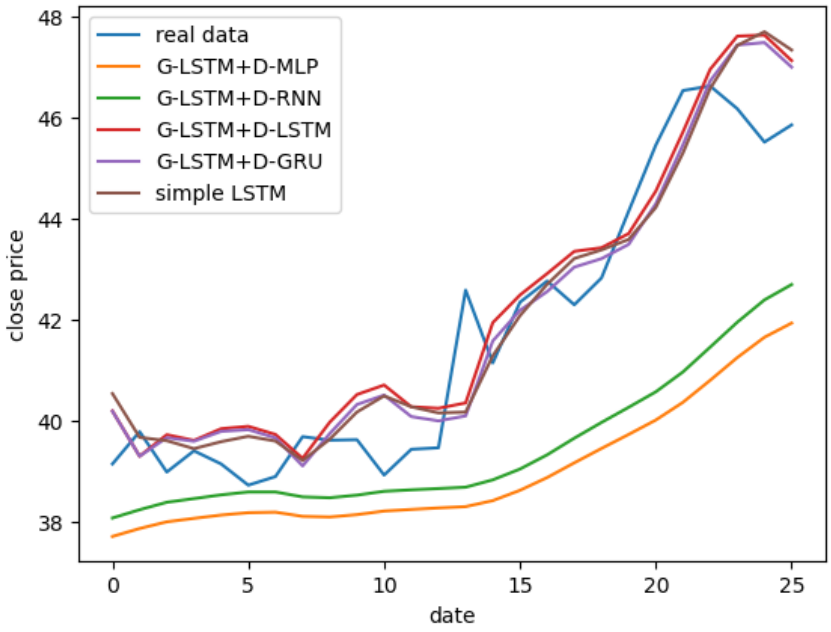

Fig 3.9: Prediction comparison using different models on test data (AMD)

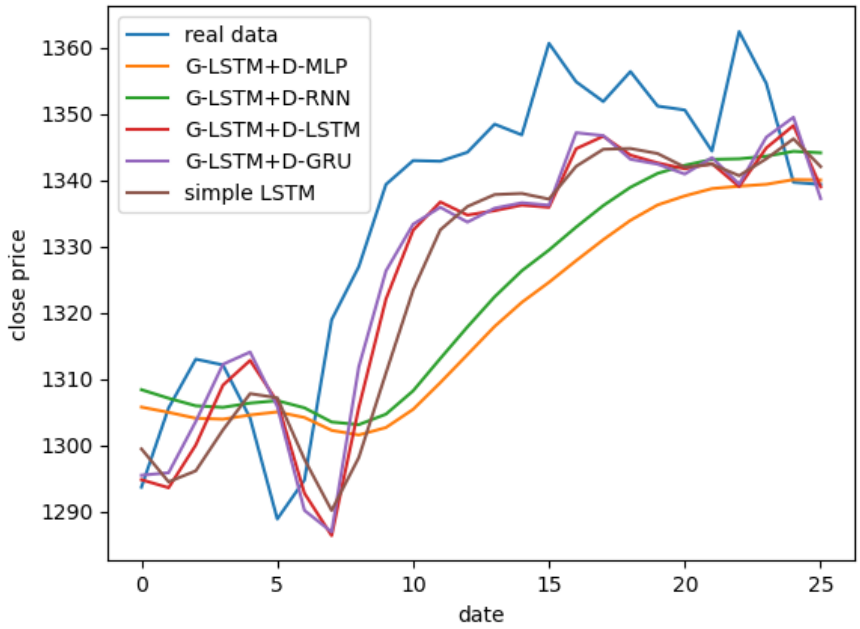

Fig 3.10: Prediction comparison using different models on test data (Google)

the G-LSTM+D-GRU model's result is worse than G-LSTM+D-LSTM, the former may be used in actual investment analysis because the latter model has much more weights to compute than the former. Also, the difference in results between the two models is not huge.

Table 3.1: Result of Training Strategy 1

model	RMSE	MAE
ARIMA	10.345	9.146
GANs	8.962	7.339
LSTM	3.727	2.822
G-LSTM+D-MLP	4.153	5.510
G-LSTM+D-RNN	4.046	4.989
G-LSTM+D-LSTM	**3.441**	**2.523**
G-LSTM+D-GRU	3.509	2.934

Strategy 2: the segmented data strategy on different model comparisons is shown in TABLE 3.2. The result of strategy 2 is beyond the scope of this research because the experiment shows the simple LSTM has the best performance. This means that compared with strategy 1, the hybrid successive GANs are more suitable for strategy 1.

Table 3.2: Result of Training Strategy 2

model	RMSE	MAE
ARIMA	17.385	15.994
GANs	14.039	13.302
LSTM	**10.145**	**8.080**
G-LSTM+D-MLP	14.908	12.045
G-LSTM+D-RNN	13.334	12.013
G-LSTM+D-LSTM	13.983	12.224
G-LSTM+D-GRU	12.348	10.125

3.4 Summary

We proposed the hybrid sequential GANs to predict the future stock close price by using three strategies. From the result of experiments we found that G-LSTM+D-LSTM model had the best performance on strategy 1, and G-LSTM+D-GRU model had the best performance on strategy 3, and beyond our expectations our model didn't perform well on strategy 2. As we show the structure in 3rd section Methodology, the RNN model has most simple structure that it can't capture the long memory information from time series dataset. So when we set RNN into the Discriminator it didn't perform well. Although the LSTM and GRU has different numbers of weights, their structures show that they have the similar effect on training the time series data. We can find this conclusion from the comparison result.

The hybrid successive GANs were proposed to predict future stock closing prices using two strategies. From the result of experiments, the G-LSTM+D-LSTM model had the best performance on strategy 1, our model didn't perform well on strategy 2. As we showed in section 2 of the Methodology section of Chapter 2, the RNN model has the simplest structure in that it cannot capture the long memory information from time series dataset. Therefore, when the RNN was set to the Discriminator, the RNN didn't perform well. Although the LSTM and GRU have different numbers of weights, their structures show that they have a similar effect on training the time series data. However, the G-LSTM+D-LSTM model performs best in strategy 1, the hyper-parameters of the model are unoptimized, which means the current result is not the best.

Chapter 4

Optimization of GANs on Stock Price Prediction

4.1 Introduction

There are two primary methodologies for forecasting financial data: fundamental analysis and technical analysis. Technical analysis shows how useful information and patterns in financial data can be extracted from statistical analysis of historical data. In recent years, the rapid development of machine learning has made this field of forecasting financial data no longer the specialization of financial scholars and economists but has gradually become an interdisciplinary subject of finance and computer science. Several algorithms have been developed for accurate analysis of financial data; for example, Convolutional Neural Networks (CNN) [17], Long Short-Term Memory (LSTM) [60], and Generative Adversarial Networks (GANs) [14]. Many studies have proved that machine learning algorithms can predict the financial data better than traditional algorithms such as the Autoregressive Integrated Moving Average (ARIMA) [56].

Among these machine learning algorithms, GANs are more flexible and work better than other algorithms. Generative Adversarial Networks (GANs) is a generative model based on the principles of game theory to build an architecture [14]. GANs have gradually become excellent generative models in time-series data generation. Several researchers have applied GANs to stock price prediction [70–72]. However, the prediction accuracy of GANs depends on the hyper-parameters and the network structure of the model. To make the model perform best, it has to be optimized. The optimization of the network architecture in the field of

CNN has been extensively studied, but because the overall structure of GANs is more complex than CNN and more difficult to train, the optimization of the GANs-based framework has become a necessary topic.

Therefore, in this work, a Genetic Algorithm (GA) [111] is applied to the design of the hyper-parameters and the network structure of the model. The Genetic algorithm (GA) is a search algorithm used for solving computational problems. GA was initially developed based on some phenomena in evolutionary biology, including heredity, mutation, and natural selection. Several previous studies focused on hyper-parameters tuning on Convolutional Neural Networks (CNN) by using GA [97,98]. GA-based optimization of GANs in stock price prediction is presented in this study. The 'chromosome' in the proposed model includes the window size of the training data and the hyper-parameters of the LSTM in the Generator.

In previous research, when using the strategy of rolling window training, the result (RMSE, MAE) has been better than the strategy of segmented training, and in strategy 1 the G-LSTM, D-LSTM combination performed best. For this reason, we optimized the successive GANs based on the G-LSTM, D-LSTM model by using the Genetic Algorithm.

4.2 Methodology

4.2.1 Successive-GANs (GANs, LSTM) – Basic Model

The combinational model of GANs and LSTM is used in this research for stock price prediction. GANs have two differentiable networks in the model. One is named the Generator, which predicts the stock price, and another is the Discriminator, which determines whether the data is real or generated by the Generator. In the original GANs, both the Generator and the Discriminator are defined by Multi-layer Perceptron (MLP). In this research, however, two different LSTM networks are used for the Generator and Discriminator because they show very good accuracy for predicting time-series data.

Figures 4.1 and 4.2 show the structure of the original GANs and the GANs proposed in this research, respectively. According to this change in GANs structure unsupervised learning is converted to a semi-supervised learning.

In addition, the window sliding method is applied to split the stock data for data pre-

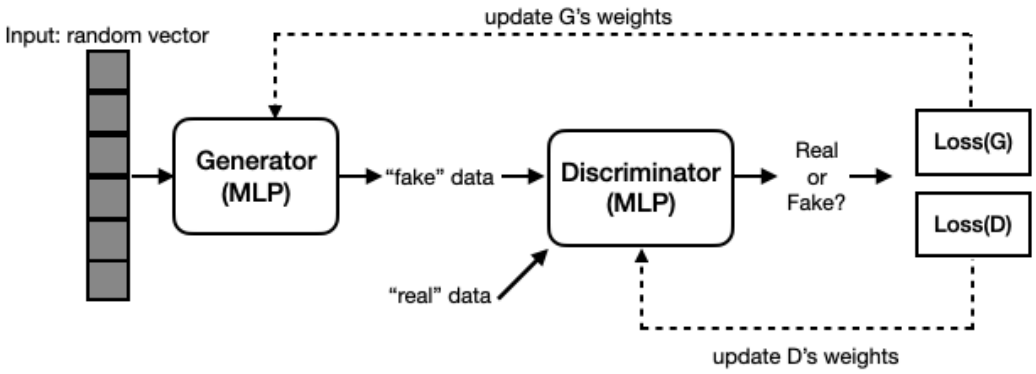

Fig 4.1: Original GANs

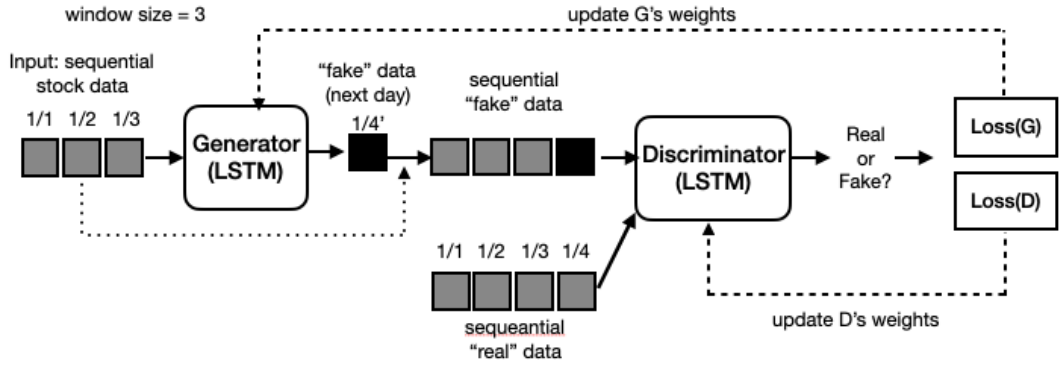

Fig 4.2: Sequential GANs

processing. Figure 4.3 shows the process of window sliding with a window size set by 3. Each number (1, 2, 3, 4, 5) indicated daily stock data of days 1, 2, 3, 4, 5, respectively. Each window includes three days of historical stock data to predict the next day closing price. This process continues until the last day of the original stock data.

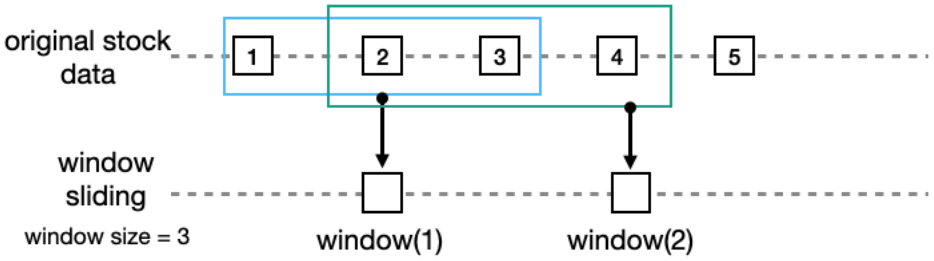

Fig 4.3: Window Sliding

4.2.2 Optimization by Genetic Algorithm

The Genetic Algorithm (GA) finds the optimal combination of hyper-parameters in GANs. In the GA, the chromosome indicated the combination of hyper-parameters. The optimization algorithm is shown in Figure 4.5. In this study, as shown in Figure 4.4, each chromosome is composed of four genes related to window size, number of layers in the Generator, number of units in the first layer of the Generator, and the number of units in the second layer of the Generator, respectively. The first gene takes values from 1 to 8; the second gene 1 or 2, and the third and fourth gene 10 to 32. If the second gene is 1 then the fourth gene is 0.

Each chromosome is defined by randomly generated numbers and the population is composed of chromosomes. Fitness functions of all chromosomes are estimated, and then genetic operators such as selection, crossover, and mutation are applied to the population. After this processing, a new population is generated. The above optimization process repeats until the best chromosome is found.

Fitness function is defined as the root mean squared error (RMSE) as follows.

$$fitness = RMSE = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(y_i - y_i')^2} \tag{4.1}$$

where the variable y_i represents the real stock data on day i, the variable y_i' represents the generated stock data and the variable m represents the number of the stock data.

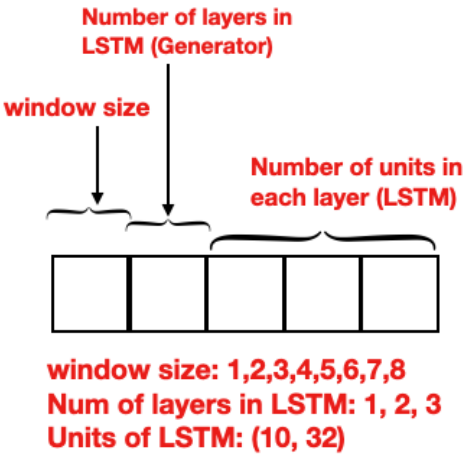

Fig 4.4: Chromosome in stock prediction

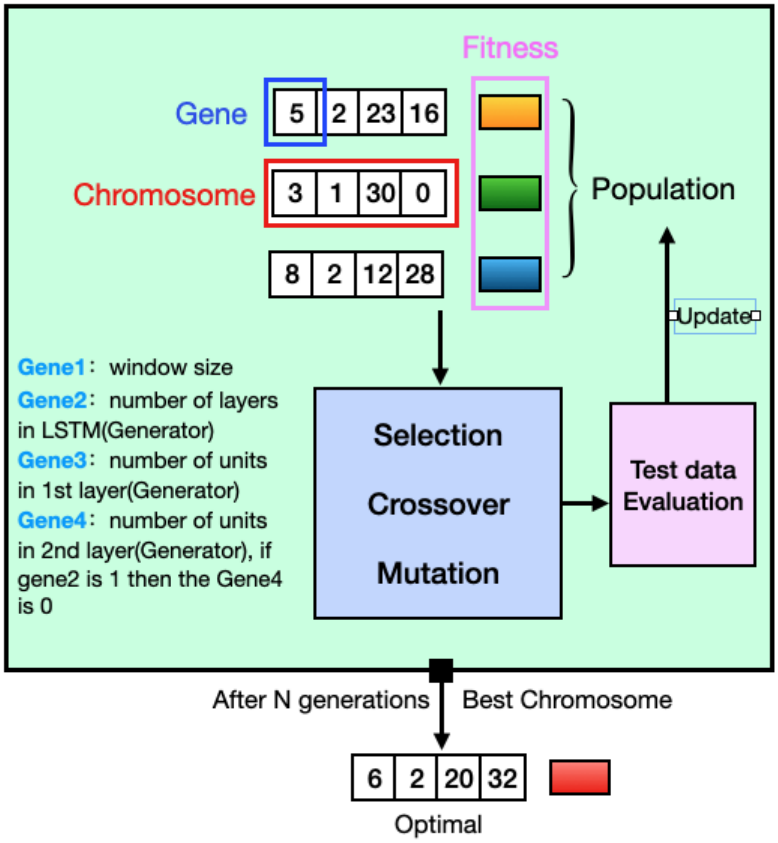

Fig 4.5: Genetic Algorithm of GANs for stock prediction

Two conditions have to be satisfied when using the GA:

a) A chromosome must be defined to represent a solution to the optimization problem. Five genes in a chromosome are set for updates in the GA operation.

b) A fitness function must be defined to evaluate the chromosomes which represent solutions. RMSE is set as the fitness for evaluation.

The three basic operations that constitute a GA, are as follows:

- **Definition of population**: The population is constructed with randomly generated chromosomes.
- **Fitness estimation**: Fitness of chromosomes is evaluated.
- **Selection**: Parents are selected according to the fitness values.
- **Crossover**: One-point crossover generates the offspring from the parents.
- **Mutation**: Offspring genes are randomly changed.

Algorithm 3 of the GA algorithm for stock prediction is represented below:

Algorithm 3 GA algorithm of GANs for stock prediction

Input:

Training data D_{train}, Testing data D_{test} ;
Number of generations T;
Population size N;
Crossover probability P_c;
Mutation probability P_m;

Output:

Final generation $\left\{M_{T,n}\right\}_{n=1}^{N}$;
Initialization: generating a set of randomized individuals $\left\{M_{0,n}\right\}_{n=1}^{N}$;
Evaluation: computing the fitness of each individual in D_{test};
for $t = 1, ..., T$ **do**
  Selection: Selecting a parent set $\left\{M'_{T,n}\right\}_{n=1}^{N}$;
  Crossover: for each pair $\left\{M_{t,2n-1}, M_{t,2n}\right\}_{n=1}^{N/2}$, performing crossover with P_c;
  Mutation: for each individual $\left\{M_{t,n}\right\}_{n=1}^{N}$, performing mutation with P_m;
  Evaluation: computing the fitness of each new individual in D_{test};
end for
Save the elite individual

4.2.3 Modified GA processing

Neural networks are not easily implemented with conventional genetic algorithms for the following reasons:

1. In the traditional genetic algorithm, the length of each chromosome is the same, but when optimizing the LSTM in the Generator, the length of the chromosome will be different because of the number of layers.

2. In the conventional genetic algorithm, the value range of the genes on the chromosome is the same, but when optimizing the LSTM in the Generator, the genes representing the number of layers need to be in a range which is different than the range of genes representing the number of units.

3. Due to the different lengths of the chromosomes, both the crossover function and the mutation function need to be modified.

To solve the above problems, the following methods are used:

1. Set each chromosome to the same length and fill in zeros if the length is insufficient. Set the second gene, make it 1 or 2, and then determine the following genes about the number of units in each layer based on the second gene.

2. For the modification of the crossover function, following the crossover point to crossover is shown in Figure 4.6 (left side). The crossover operation stops when the second gene is different between two chromosomes.

3. For the modification of the mutation, only the gene is mutated where it is the window size and the number of units, and the mutation on the gene stops, which is the number of layers as shown in Figure 4.6 (right side).

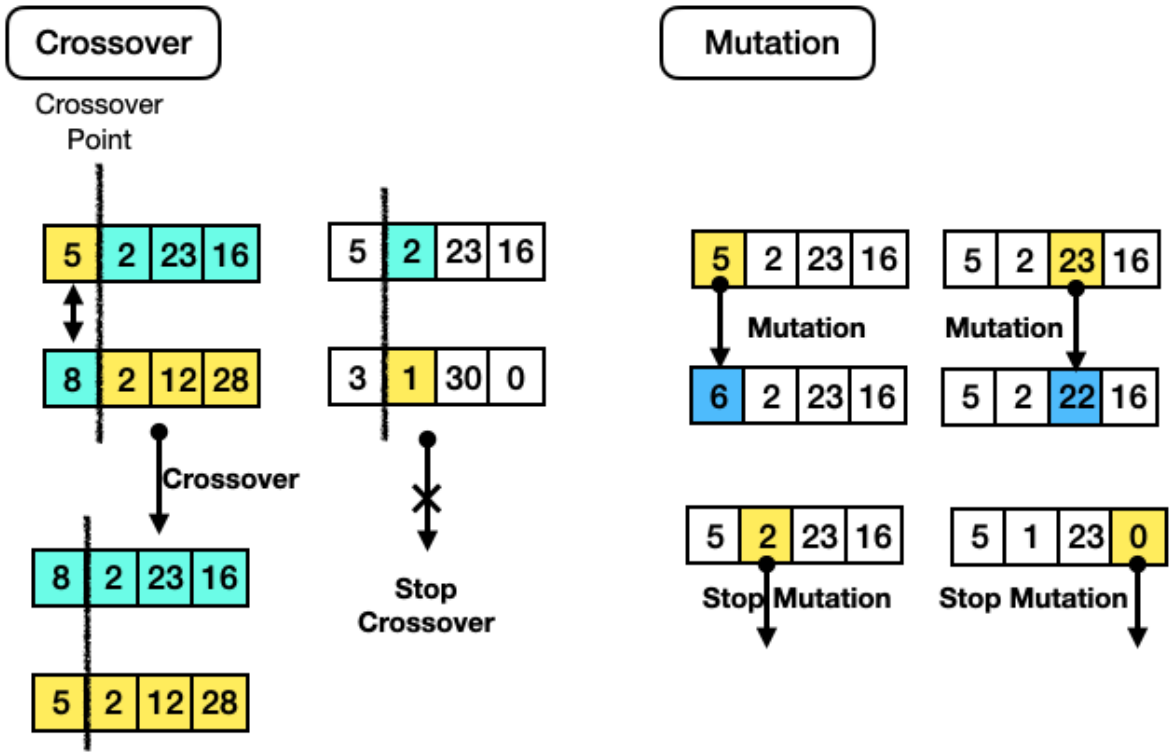

Fig 4.6: Modified crossover and mutation operation

4.3 Experiment

4.3.1 Experimental Dataset

In this work data from 4 companies are used: Apple (AAPL), Advanced Micro Device (AMD), Adobe (ADBE), and Google (GOOGL) for training. The training data are from 2010/01/01 to 2019/11/30, the test data are from 2019/12/01 to 2019/12/31.

4.3.2 Pretest for GA design

One-tenth of the training data was used as a pre-training to test the crossover rate and mutation rate of the Genetic Algorithm. First, the population was set to 20 and the generation to 20. The results after the change of crossover rate and variation rate are compared in the Table 4.1. "CR" stands for crossover rate and "CM" stands for mutation rate. From the Table 4.1, it shows that the best result is obtained when the cross rate is 0.6 and the variation rate is 0.1.

Table 4.1: Effect of crossover and mutation rates (stock)

CM \ CR	0.4	0.5	0.6
0.1	13.023	12.883	**11.930**
0.2	13.924	14.057	12.304
0.3	12.792	13.472	12.770

Although in many optimization problems population and generation are as large as possible, it will waste a lot of computational resources and be inefficient if they are set too large. Therefore, in this study, in order to select the appropriate population and generation, two indicators were set. For the setting of generation, if the last 5 generations' change is less than 1%, the result is "SA", which means satisfied; more than 1%, the result is "UN", which means unsatisfied. Only the combination of generation and population which have the "SA" result can be selected. For population, the average fitness will calculate for selection.

Secondly, due to the Table 4.2 and the above explanation, the population as 20 and the generation as 40 had been chosen. In summary, the parameters of GA are crossover rate = 0.6; mutation rate = 0.1; population = 20; generation = 40.

In Figure 4.7, here fitness is defined by RSME. x-axis is generation and y-axis is fitness.

Table 4.2: Selection of population and generation (stock)

Population \ Generation	10	20	30	40
10	UN/12.149	UN/11.382	UN/11.039	SA/10.801
20	UN/11.802	UN/11.142	UN/10.991	**SA/10.553**

we can see from the graph that as the genetic algorithm is trained fitness gets smaller, which means that better and better individuals are selected. At about 30 generations, we can see that the selection of the best individuals has basically stopped. This means that there is no more training value to increase the generation.

In the Table 4.3, the optimal hyper-parameters of successive-GANs model are shown.

Table 4.3: optimal hyper-parameters

company	window size	dense	units (1st)	units (2nd)	units (3rd)
AAPL	5	3	20	22	13
AMD	6	2	19	15	0
ADBE	6	3	30	20	12
GOOGL	5	2	30	16	0
fixed1	5	1	32	0	0
fixed2	5	2	32	20	0
fixed3	5	3	32	20	16

4.3.3 Experimental Results

In Figures 4.8, 4.9, 4.10, and 4.11, and in Table 4.4 and Table 4.3 the result are compared with the sets of initial and optimized hyper-parameters. The Generator predicts the future stock data and the Discriminator is abandoned. In Figures 4.8, 4.9, 4.10, and 4.11, and in Table 4.4, the x-axis is the data from 2019/12/01 12/31, and the y-axis is the stock price of each company. In the four companies (Apple, AMD, Adobe, Google), the result of optimized sequential GANs by GA is better than fixed GANs. In Tables 4.4 and 4.3, the optimized model is compared with the fixed model, which includes the number of layers are 1, 2, and 3. From the result of the stocks from Apple and Adobe, the optimal number of layers is 2, which means the number of layers is not better than when using GANs to predict future stock prices. The results show that our initial setting cannot acquire the best predict accuracy. Therefore, the

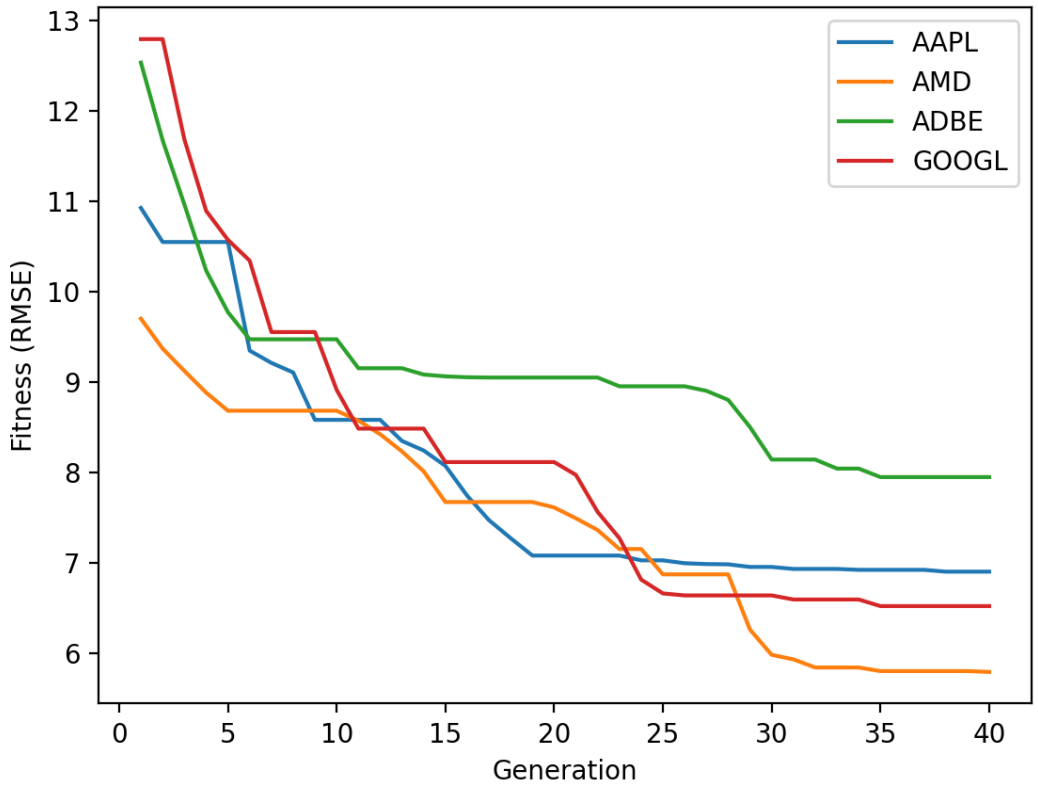

Fig 4.7: Convergence of the best individual fitness with respect to the generation number. (stock)

design of model structure is not reliable. Although the accuracy on stock prediction by GA is used, the training time's cost is about 20 times more than in the fixed model.

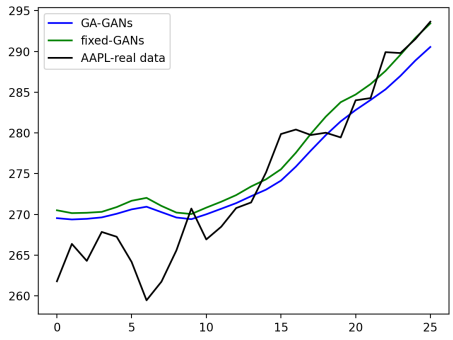

Fig 4.8: Apple

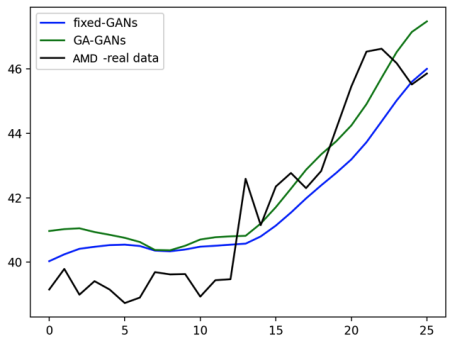

Fig 4.9: AMD

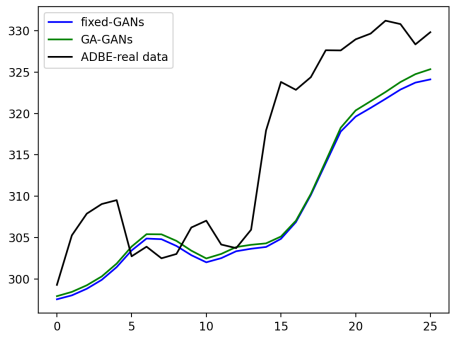

Fig 4.10: Adobe

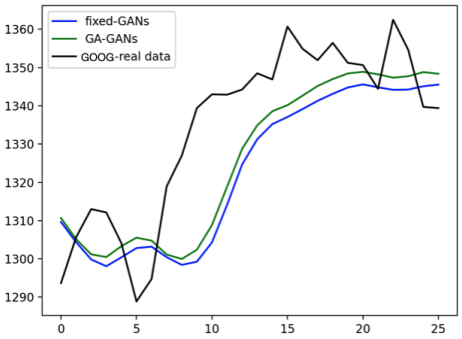

Fig 4.11: Google

4.4 Summary

In this study, the GA-based optimization approach is applied for the design of hyper-parameters of Generative Adversarial Networks on the stock price prediction. Through the operations inside with GA, the best hyper-parameters are acquired in combination in the GANs model for each company. The experimental results show that the performance of GANs model has been improved after optimization. In addition, the know-how and experience of machine learning designers are no longer necessary when using hyper-parameters tuning by GA.

Table 4.4: Compare with two fixed GANs

company	RMSE(Initial-GAN1)	RMSE(Initial-GAN2)	RMSE(Initial-GAN3)	RMSE(GA-GAN)
AAPL	7.112	7.082	6.951	**6.901**
AMD	5.982	5.991	5.837	**5.791**
ADBE	8.027	8.007	7.970	**7.946**
GOOGL	6.759	6.810	6.661	**6.519**

Chapter 5

Successive Image Generation of Pedestrian Walking Behavior

5.1 Introduction

When a car approaches a pedestrian, the car driver must judge whether the pedestrian will cross in front of the car so as to avoid hitting the pedestrian. If a system can predict the pedestrian's future behavior, the vehicle can recognize the existence of the pedestrian in front of the car and the walking direction of the pedestrian. This will avoid traffic accidents with pedestrians. In previous studies, pedestrian motion patterns or what is called, the moving path, is predicted by the Bayesian model and scenario patterns. This study generates the pedestrian's future images from past images which are taken through a camera.

The proposed algorithm generates the pedestrian's future successive images from their past successive behavior images. Algorithms for generating successive images have been studied widely. The previous algorithms are based on Optical flow or Long Short-term Memory (LSTM). On the other hand, an algorithm using Generative Adversarial Networks (GANs) is employed in this study. Generative Adversarial Networks (GANs) were presented by Goodfellow in 2014. GANs are classified into unsupervised machine learning algorithms. GANs have outstanding performance in the field of image generation.

There are several advantages while using the GANs model:

- GANs can generated the realistic-looking images. Therefore, researchers can acquire lots of worthwhile information from the predicted future images.
- Previous research has shown, however, that predicting a pedestrian's future move-

ments when she/he suddenly turns direction is difficult. The GANs model can predict future movement by increasing the diversity of pedestrian's data sets.

- The GANs model can also predict when suspicious people are near the pedestrian.

The proposed algorithm is defined as the concept of Generative Adversarial Networks (GANs). While the original GANs use fully-connected layers for the Discriminator, the proposed algorithm adopts Convolutional Neural Network for the Discriminator. For numerical experiments, successive images of pedestrians are necessary. Microsoft KINECT [115], or, KINECT, is used for recording pedestrians' walking scenes. Segmenting the video of the pedestrians creates successive images. In numerical experiments, while the number of pedestrians' past images is taken as 2, 4, 6, 8 and 10, the number of pedestrians' future images varies 2, 4, 6, and 8. The accuracy is defined as the difference between the real and the generated future images. The proposed algorithm is compared with Optical flow to confirm the validity of the proposed algorithm.

5.2 Methodology

The proposed method is called successive-GANs [116]. There are several differences between the original GANs and successive-GANs. The difference in the Generators of both GANs is summarized as follows. The input of the original GANs Generator is random noise while the input of successive-GANs are continuous images data. The difference in the Discriminator is that the input of the original GANs is a "real" image and the "fake" image, which is from the Generator, is output on a scale between 0 and 1. When the scale is close to 0, 0 represents the Discriminator's 'belief' that the image was "fake". On the contrary, when the scale is close to 1, this represents the Discriminator's belief the image was "real". The input of successive-GANs has two sets; one set includes the Generator input images data and generated images, the other set includes the same Generator input images and ground truth images. Ground truth images are the images that were supposed to be predicted.

In the successive-GANs model, Generator G first takes several continuous pictures X, and then uses the Convolutional Neural Network to generate one future image y'. Generator G has to minimize the distance $L(y, y')$ between the generated picture y' and the ground

truth y in order to train the network using back propagation.

$$L(y, y') = L(y, G(X)) = \left\| y - G(X) \right\| \tag{5.1}$$

The Discriminator needs to replace the above loss function. That means the Discriminator's loss function is used by the Generator changing the weight of a back propagation to make the generated image more "real".

Training Discriminator

Let (X, y) be a continuous sample from the dataset. Let $G(X)$ be one image generated by the Generator. The input of Discriminator is (X, y) and $(X, G(X))$, and the output is a scalar in the range between 0 and 1. In this way, the discriminator tries to classify the input (X, y) into class 1 and the input $(X, G(X))$ into class 0. Therefore, the Discriminator is more like a binary classification CNN. When the Discriminator is training, the weights of the Generator have no change, which means that Generator is not training in the meanwhile. The difference between the original GANs and the successive-GANs in the Discriminator is in the number of input images.

Therefore, the loss function for training the Discriminator is:

$$L^D(X, y) = L_{bce}(D(X, y), 1) + L_{bce}(D(X, G(X)), 0) \tag{5.2}$$

where L_{bce} is the binary cross-entropy loss function, which is defined as

$$L_{bce}(Y, Y') = -\sum_i Y'_i \log(Y_i) + (1 - Y'_i) \log(1 - Y_i) \tag{5.3}$$

Minimizing this loss function means that the ability of the Discriminator in discriminating the "real" data from the "fake" data has been improved.

Training Generator

The input of the Generator is a different "real" continuous sample X, and the output is one image $G(X)$ that generated by the Generator. While fixing the weights of the Discriminator, the adversarial loss defined below should be minimized:

$$L^G(X, y) = L_{bce}(D(X, G(X)), 1) \tag{5.4}$$

Minimizing this loss function means that the ability of Generator in cheating the Discriminator has been improved. In other words, image generated by Generator is more like the "real" data.

Process

The algorithmic process is summarized as follows:

1. The weights of the Generator are fixed and the Discriminator is trained to minimize the loss function to improve its ability to discriminate the "real" from the "fake" data. This process can update the weights of the Discriminator using back propagation of loss function.

2. The weights of the Discriminator are trained to train the Generator by using the same loss function. This process can update the weights of the Generator by using back propagation of the same loss function.

3. Steps 1) and 2) are iterated in the Algorithm 4 as shown below: W_D and W_G denote the weights of the Discriminator and the weights of the Generator, respectively.

Both the Generator and the Discriminator are neural networks. In the work [88], fully connected layers are used in the Discriminator [108]. In this thesis, only one pedestrian is in a frame, and the background has no information. Therefore, the fully connected layers (FC) are turned into convolutional layers because the FC layers have two faults [47]:

- A fatal weakness in the FC layer is that the parameter size is too large, especially the FC layer connected to the last convolutional layer. In the interval at the same time, the calculation amount of training and testing is increased, and the calculating speed is reduced.

- The parameter amount is too large to be over-fitting of the training model. Although the parameter uses dropout, dropout is a hyper-parameter, so it has to be set by researchers.

Two advantages change fully connected layers to convolutional layers:

Algorithm 4 Improved GANs algorithm

Input Data:
continuous image data M: $(X^{(1)}, y^{(1)}), \cdots, (X^{(M)}, y^{(M)})$
Generator:

- input: $X^{(1)}, \cdots, X^{(M)}$
- output: $G(X^{(1)}), \cdots, G(X^{(M)})$
- Generator learning rate: l_G

Discriminator:

- input: $(X^{(M)}, y^{(M)})$ and $(X^{(M)}, G(X^{(M)}))$
- output: a scalar value between 0 and 1
- Discriminator's learning rate: l_D

for number of training iterations EPOCH **do**
One data sample is obtained from M, the Generator is fixed, and the Discriminator updates weights W_D:

$$W_D = W_D - l_D \sum_{i=1}^{M} (\frac{\partial L^D(X^{(i)}, y^{(i)})}{\partial W_D} + \frac{\partial L^D(X^{(i)}, G(X^{(i)}))}{\partial W_D})$$

In another data sample from M, the Discriminator is fixed, and the Generator updates weights G_D:

$$W_G = W_G - l_G \sum_{i=1}^{M} \frac{\partial L^G(X^{(i)}, y^{(i)})}{\partial W_G}$$

end for

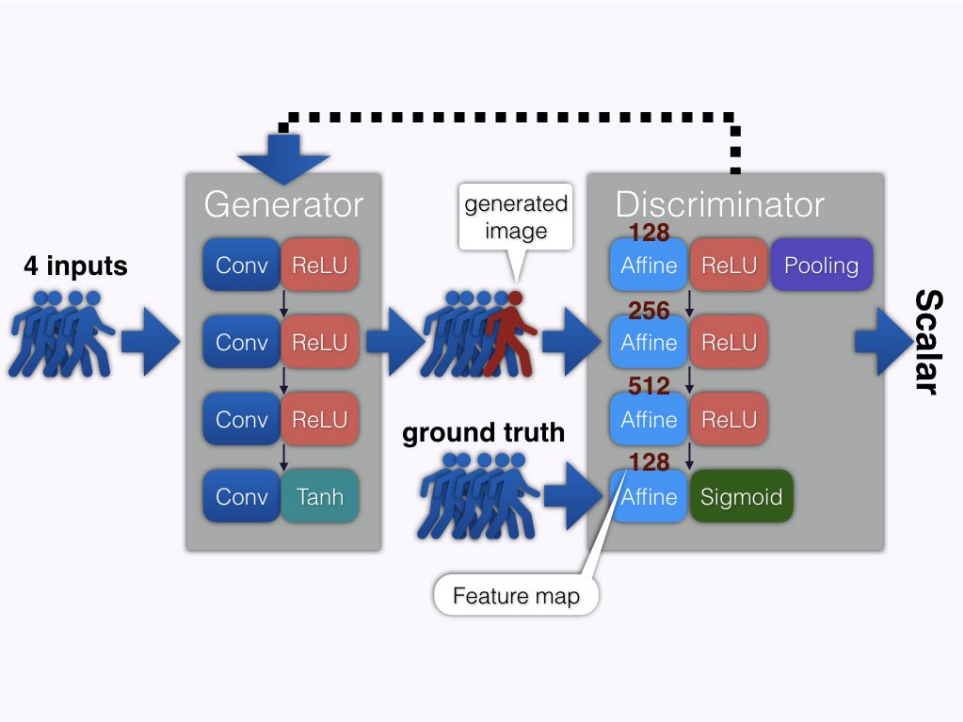

Fig 5.1: GANs with Fully connected layers (before)

- The convolutional layer has a local connection; the fully connected layer uses the global information of the image. In this research, the whole image is not calculated because the information needed exist only in the pedestrian's part.

- This change allows the convolutional network to slide over a larger input picture to get the output of each area (so that it breaks the input size limit).

This change shows the difference changed in the structure inside the successive-GANs (Figure 5.1 and 5.2).

Next, a flowchart of the experiments performed in this research using GANs is shown below:

Flowchart:

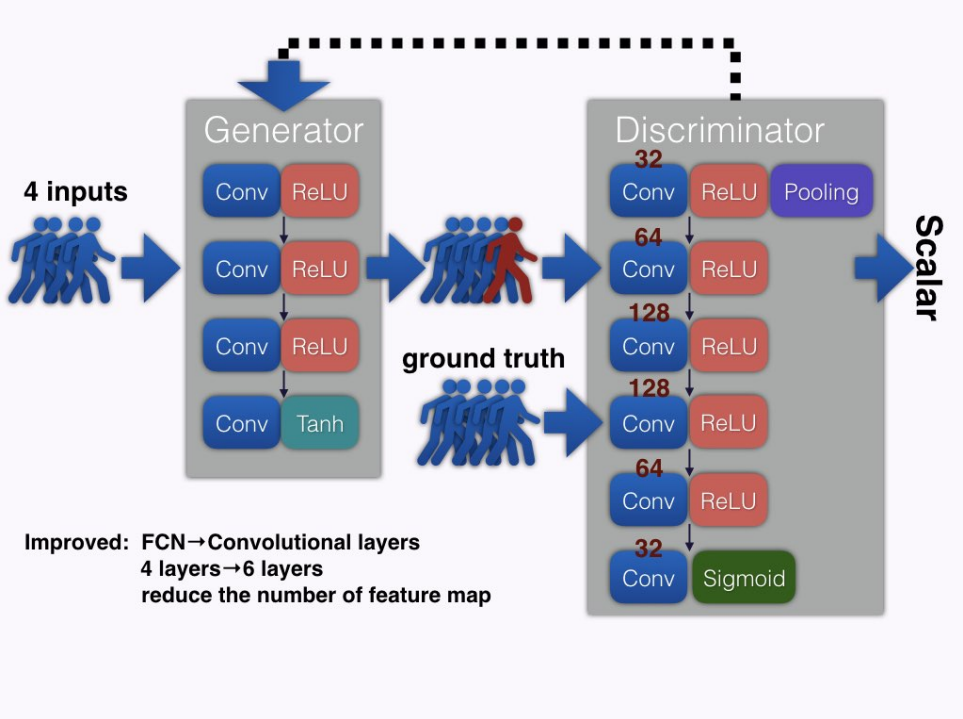

Fig 5.2: GAN with Convolutional layers (after)

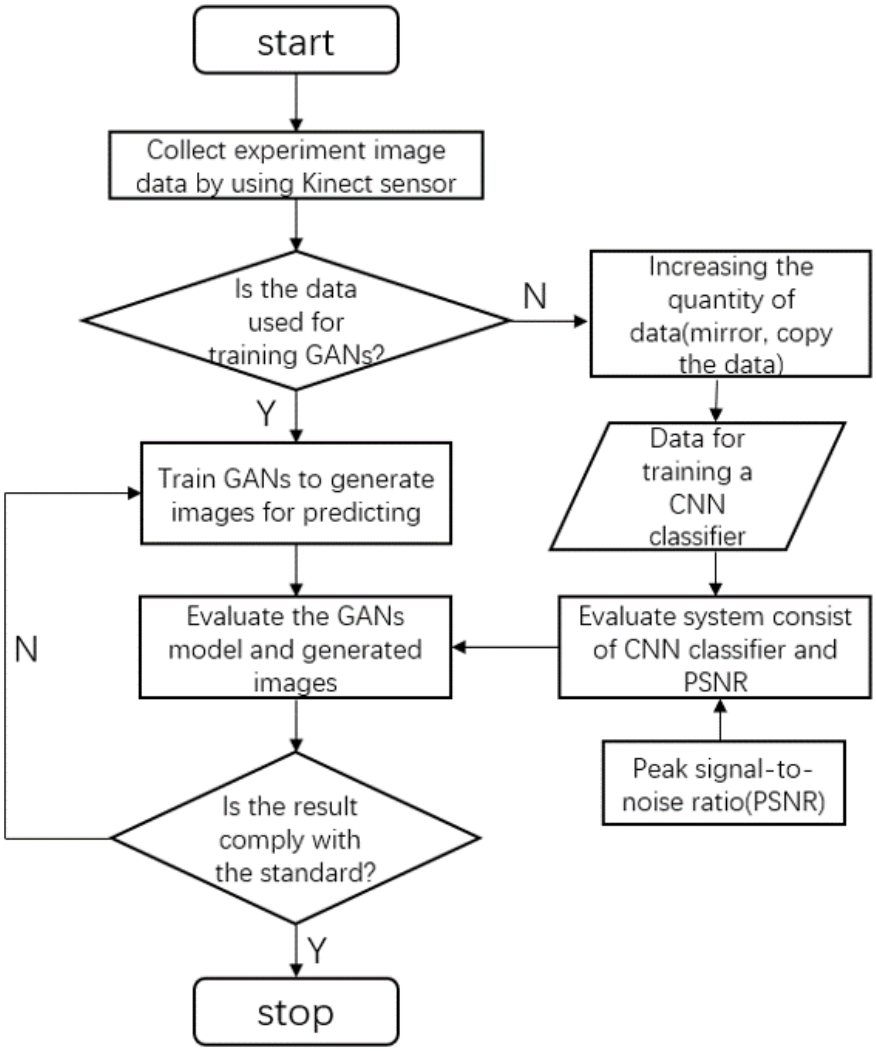

Fig 5.3: Flowchart of proposal GANs

5.3 Experiment

5.3.1 Datasets

A pedestrian dataset for this research used the RGB camera of Kinect. The dataset included five people. The Kinect was set at the height of 2 meters, because the Kinect can capture people's movements completely at this height. Many surveillance cameras were also set to this height. The goal was to simulate a scene where the pedestrian was walking in a public area. Next, each pedestrian's video was captured in 8 directions (Figure 5.4). Each video has 25fps; therefore, a 1-second video has 25 frames.

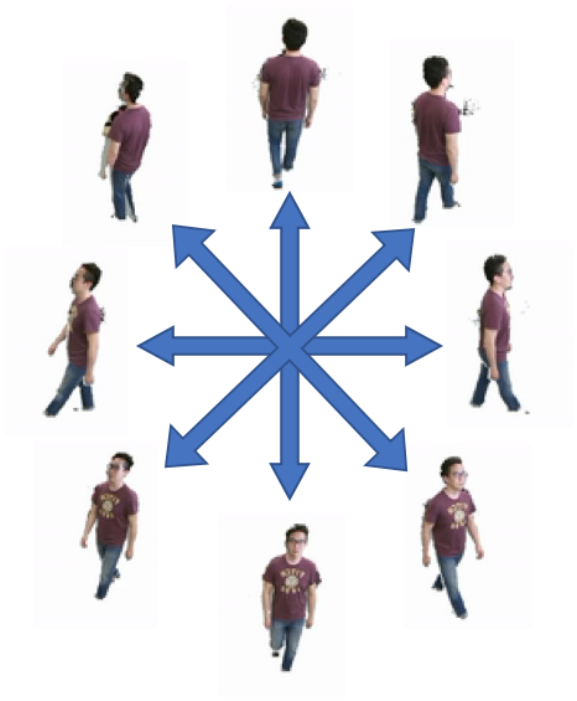

Fig 5.4: 8-directional data

The pre-processing of data is performed as follows: In the improved GANs, the number of input frames and the number of output frames are important hyper-parameters. The data collected was 25fps video. Therefore, several frames had to be extracted from one video data, which means that training photo data was trained. The experiments were designed in several pairs as follows: 2-2, 4-2, 4-4, 6-2, 6-4, 6-6, 8-2, 8-4, 8-6, 8-8, 10-2, 10-4, 10-6, 10-8, 10-10 (the front number is the input frame's number and the back number represents the output number).

In the evaluation of CNN, a CNN was trained to evaluate the GANs. In one direction, 10 frames were extracted from one video, and 5 videos were taken in one direction. Next, OpenCV was used to increase the quantity of the dataset, including removing the background (Figure 5.5), rotation, and to make mirror frames. After this process, 10 frames were transferred to 40 frames. In total, 40*5*8*5=8000 frames could train. The first 5 frames means that in one direction 5 videos were taken, and 8 means 8 directions. In the latter 5 there is walking data from 5 pedestrians. Next, the 8000 frames were separated into 2 parts: training data and testing data; the proportion was 8:2. The purpose of this process was to evaluate whether or not the CNN can judge if the pedestrian frame is predicted and generated by improved GANs, in one of 8 directions.

Fig 5.5: Removal of Background of Picture

5.3.2 Experimental Details

The experiment is aimed at generating several continuous images as the prediction of pedestrian behavior using the dataset mentioned in the last section. The experimental code is written by python3.6 in this work. The python package used frequently in this experiment included tensorflow, numpy, skimage, and scipy. Both GTX 1060g and GTX 730 GPU are used in this experiment. Though GPUs reduce a lot of computing time, it takes several minutes to train GANs networks on 100steps. Before training the networks, several hyper-parameters must be set; for example, the number of the training, the number of the input images, the number of the output images (predicted images), the Generator learning rate, and the Discriminator learning rate. When the number of training steps reaches the number of the training, one experiment is finished.

5.3.2.1 Evaluation by PSNR This thesis uses two evaluation methods. The first method is the Peak signal-to-noise ratio (PSNR) [117]. PSNR is an engineering term for the ratio between the maximum possible power of a signal and the power of corrupting noise that affects the fidelity of its representation. PSNR is most used to measure the quality of reconstruction of lossy compression codecs (for example, for image compression). The signal, in this case, is the original data, and the noise is the error introduced by compression. When comparing compression codecs, PSNR approximates human perception of reconstruction quality. Different from the reconstruction situation, in this thesis the generated frames and the ground truth frames with PSNR are examined. PSNR is often simply defined by mean square error (MSE). For two m*n monochrome images I and K, then their mean square error is defined as:

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i,j) - K(i,j)]^2 \tag{5.5}$$

The peak signal to noise ratio is defined as:

$$PSNR = 10 \cdot \log_{10}(\frac{MAX_I^2}{MSE}) = 20 \cdot \log_{10}(\frac{MAX_I}{\sqrt{MSE}}) \tag{5.6}$$

Between them, MAX_I is the maximum value indicating the color of the image point. If each sample point is represented by 8 bits, MAX_I is 255. A more general representation shows that if a 8-bit linear pulse code modulation represents each sample point then MAX_I is

$$MAX_I = 2^8 - 1 \tag{5.7}$$

The definition of the peak signal-to-noise ratio is similar for a color image with three values of RGB per point, except that the mean square error is the sum of all variances divided by the image size divided by three.

Typical PSNR values in image compression range from 30 to 40 dB, and the higher the value, the better.

5.3.2.2 Evaluation by using a CNN classifier Generally, however, the PSNR has performed poorly compared to other quality metrics when estimating the quality of images and particularly videos as perceived by humans. For this reason, this research used an additional second evaluation method; namely, the 8000 images data to train a CNN classifier model to work in 8 directions. Evaluating and comparing GANs, or evaluating and comparing images produced by GANs, is a very challenging task, in part because of the lack of explicit likelihood methods commonly used in comparable probability models. Therefore, many previous works on images synthesized by GANs used only subjective visual assessment. The current best GANs generated image sample cannot be evaluated accurately for image quality using subjective evaluation methods. The evaluation method used in the research is a CNN classifier, which can test the generated images quality objectively, and let researchers know which direction the pedestrian will go (Figure 5.6).

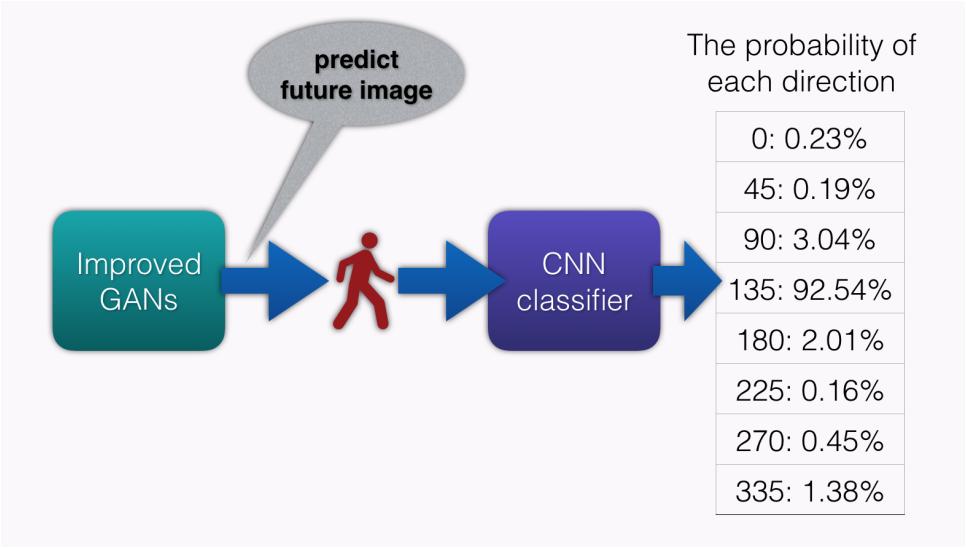

Fig 5.6: CNN classifier2

5.3.3 Experimental Results

Learning rate

Deep learning uses a straightforward first-order convergence algorithm called the gradient descent method. No matter how many adaptive optimization algorithms are used, these

algorithms are essentially a variety of deformations of the gradient descent method, so the initial learning rate, which converges to the deep network, plays a decisive role. However, in the proposed GANs there are two networks, which means there are two learning rates. The Generator was used for training the network, and then compared the Discriminator's loss function in a different learning rate. Then the Discriminator was fixed for the training network, compared with the Generator's learning rate. This experiment done with four frames of input and two frames of output. When the Generator's learning rate is too small, it will make both the Generator's and Discriminator's loss function worse. Therefore many experiments were performed with different learning rates. The G learning rate=0.00004 and the D learning rate= 0.02, so that c the Generator loss and Discriminator loss was less.

Training number

Generally, when the improved GANs training is unstable, the result is abysmal. When the training number became very large, improved GANs had the possibility of collapsing and the generated image was not good.

The specific reasons of the above problem can be explained as follows: improved GANs uses the method of confrontation training, and Generator's gradient update comes from the Discriminator, so the Generator's training is determined by the Discriminator. Specifically, the Generator generates a sample image and gives it to the Discriminator to judge if it is real or fake. The Discriminator will output the probability that the image is a true sample (0-1), which is equivalent to telling the Generator how much the authenticity of the sample is generated. The Generator will use this authenticity as feedback to adjust its parameters. The Generator constantly improves itself and increases the probability of the Discriminator's output. However, if a Generator generates a sample image that may not be very realistic, but the Discriminator gives a high scalar of evaluation. Then Generator will consider it's output is correct. In this situation, the Generator is not very good, but the Generator and Discriminator are deceiving themselves, resulting in missing information and incomplete features.

Due to the above reasons, the appropriate training number must be found. In a 20000

Table 5.1: Results

Input and output number	Optical Flow	successive-GANs	The accuracy of CNN classifier
2-2	27.8	25.2	96.6%
4-2	30.2	27.5	94.8%
4-4	28.4	26.9	95.4%
6-2	27.6	28.2	96.8%
6-4	25.8	27.4	93.0%
6-6	23.1	25.5	95.9%
8-2	27.0	**29.1**	**98.3%**
8-4	26.7	27.4	**98.0%**
8-6	23.3	26.0	96.8%
8-8	21.4	24.7	95.1%
10-2	27.0	26.3	**98.5%**
10-4	24.9	25.0	97.7%
10-6	23.4	24.9	95.5%
10-8	21.9	24.5	93.0%

training number, the improved GANs has the most stable loss in the Generator and Discriminator. In this experiment, four frames were used for input and two frames for output.

Find the most suitable numbers of input frames and output frames

The reason that the appropriate numbers of input frames and output frames have to be found in this research: The experiments showed that if the number of input frames and output frames were two big, the generated frames were crushed and the Discriminator's and Generator's loss function becomes strange. Therefore only the experiments can find the most appropriate numbers of input and output frames. The pairs in the experiments are as follows: 2-2, 4-2, 4-4, 6-2, 6-4, 6-6, 8-2, 8-4, 8-6, 8-8, 10-2, 10-4, 10-6, and 10-8. The first number is input frame's number, and the second number represents the output number. Comparing with optical flow method [118], the most suitable experimental numbers were found by comparing the PSNR scalar and the accuracy of CNN classifier (Table 5.1).

More images that improved GANs predicted means that future information can get from the generated images, so except for comparing the PSNR and the accuracy of the CNN clas-

sifier of generated images, more images should be generated. The generated images: (8 inputs and 2 outputs) are shown below. Figures 5.7, 5.8, 5.9 are the experimental images.

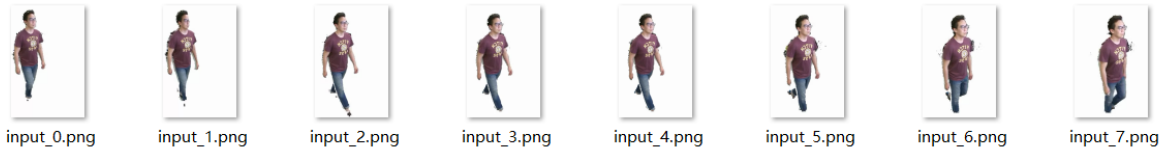

Fig 5.7: 8-input

Fig 5.8: 2-output

Fig 5.9: 2-groundTruth

5.4 Summary

In this research, a successive GANs model was trained to predict pedestrian behavior. GANs are the unsupervised learning of deep learning areas. Two methods were proposed to evaluate the model: PSNR and a CNN classifier. The successive GANs algorithm is improved from fully connected layers to convolutional layers to make the calculation more efficient and decrease the loss function. The dataset in the research is original, and the quantity of dataset is increased to train the CNN. Collecting data is a complicated job, and it is expensive in terms of time. That is one important reason why GANs are used, which is an unsupervised machine learning algorithm. When a GANs model should be trained, a small quantity of data will be enough. Also, the labeled data is difficult to obtain.

However, there are still many unresolved issues in this study. For example, the input and output image data had been set manually in this study, and these numbers need to be set in an automatic data pre-processing way, because manual setting has some involuntary operation and poor interpretability. In addition, the hyper-parameters in the GANs model are also set manually, which also needs to be automated in some way.

Chapter 6

Optimization of GANs on Successive Image Generation of Pedestrian Walking Behavior

6.1 Introduction

Future video prediction is considered a promising application for unsupervised feature learning using Generative Adversarial Networks (GANs). The input video data can be regarded as successive images. In other words, future video prediction will require the analysis and prediction of time series images analyzed by GANs and CNN. In this training process, the number of input images and the number of generated images are important. In Chapter 5, hyper-parameters such as the number of input images and output images must be set by researchers. This means that an adequate selection of the hyper-parameters and the network structure often depends on the researchers' know-how and experience. To tune the hyper-parameters automatically instead of with human assistance, in this research, an approach to optimize the number of input images, the number of generated images and the active function of generator by using Genetic Algorithm (GA) based on this composite model was proposed. Before the optimization process, the best set of the numbers of input and output images were found to be 8 and 2, respectively. After the optimization by GA, the result showed that the 8-1 combination and ReLU-tanh-ReLU-tanh in the Generator have the best performance; in other words, the best PSNR score.

6.2 Methodology

6.2.1 Basic Model

In this model, different CNN structures in the Generator and Discriminator were set. Instead of the random vector input of original GANs, successive images as the input of Generator were set. The Discriminator distinguished the real successive data from the generated data.

6.2.2 Optimization Process

Activation functions are an essential part of the design of a GANs model. The selection of an activation function in the hidden layer may connect to the performance of learning the training dataset. The basic theory of deep learning is based on an artificial neural network. Because of the repeated superposition of these non-linear functions the neural network has enough capacity to capture complex patterns and achieve state-of-the-art results in various fields. Obviously, the activation function plays an important role in deep learning, and this function is also a critical research field. The selection of the activation function has a huge impact on the performance and capability of the model, and different activation functions can be used in different parts of the designed model. Generally, in a differentiable neural network, all hidden layers use the same activation function, and the output layer uses different activation function with hidden layers.

The researchers usually use different activation functions in different types of networks:

- Multiplayer Perceptron (MLP): ReLU activation function
- Convolutional Neural Network (CNN): ReLU activation function
- Recurrent Neural Network (RNN): Tanh and/or Sigmoid activation function

However, in different GANs models, there are different activation functions. The selection of the activation function is determined by the researchers' know-how and experience. Therefore, in this research, the activation function was designed into a part of the chromosome using the GA optimization approach.

Next, GA was used to optimize the best combination of input images' number, generated images' number, and active function in the Generator. The chromosome is shown in Figure 6.1.

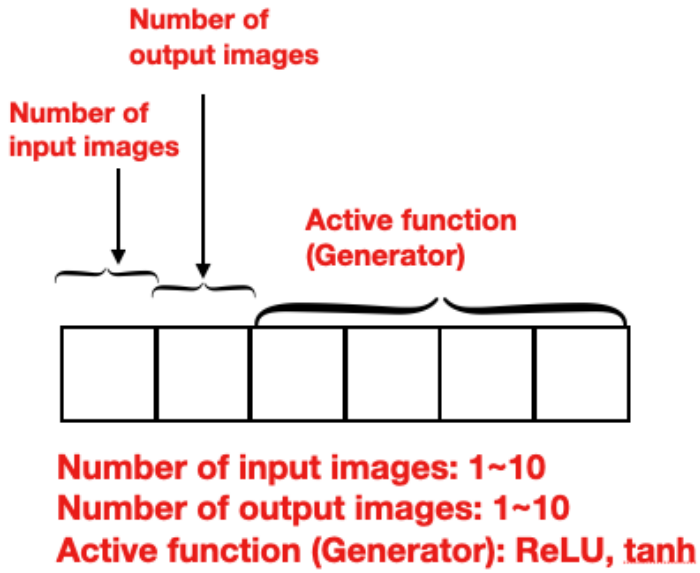

Fig 6.1: chromosome (image)

Using the Genetic Algorithm to optimize GANs on successive images is different from numerical data. For continuous images (video), the composition of chromosomes includes the number of input continuous images. At the same time, the number of continuous output images, which is generated by the GANs, is also important, because the larger the time span of the generated images is, the worse the quality of the images becomes. Therefore, the number of output images is also added to the chromosome, to achieve the best quality model. Secondly, when training continuous images, since the Generator uses CNN, the activation function is relatively important. In this thesis, the activation function was connected after the 4-layer convolutional layer as part of the chromosome. Figure 6.2 shows the optimization process of sequential GANs on successive images.

6.3 Experiment

6.3.1 Experimental Dataset

The training data was created by KINECT. The walking behavior of the pedestrian video data has 8 directions. First, the video data was turned into successive images data. Next, the background of the pedestrian image was removed, and mirror image processing was used for the data augmentation. PSNR was used to evaluate the generated images for compari-

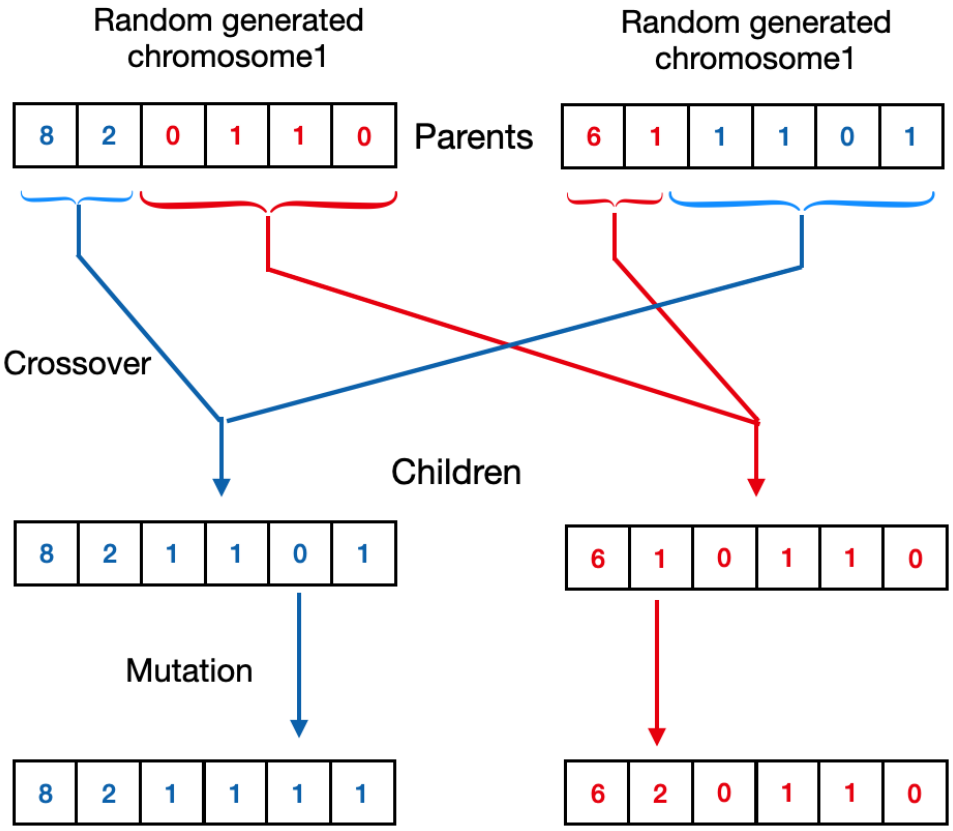

Fig 6.2: GA on successive image data

son.

6.3.2 Pretest for GA design

One-tenth of the training data was used as a pre-training to test the crossover rate and mutation rate of the Genetic Algorithm. First, the population was set to 20 and the generation to 20. the results after the change of crossover rate and variation rate are compared in the Table 6.1. "CR" stands for crossover rate and "CM" stands for mutation rate. From the table 6.1, we can see that the best result is obtained when the cross rate is 0.5 and the variation rate is 0.1.

Table 6.1: Effect of crossover and mutation rates (image)

CM \ CR	0.4	0.5	0.6
0.1	21.9	**23.7**	22.3
0.2	20.5	22.5	21.7
0.3	20.2	22.5	20.9

Although in many optimization problems population and generation are as large as possible, it will waste a lot of computational resources and inefficient if they are set too large. Therefore, in this study, in order to select the appropriate population and generation, two indicators were set. For the setting of generation, if the last 5 generations' change is less than 1%, the result is "SA", which means satisfied; more than 1%, the result is "UN", which means unsatisfied. For population, the average fitness will calculate for selection. Only the combination of generation and population which have the "SA" result can be selected.

Table 6.2: Selection of population and generation (image)

Population \ Generation	10	20	30	40
10	UN/21.1	UN/23.4	SA/25.2	SA/26.1
20	UN/22.6	UN/23.7	**SA/26.8**	SA/26.7

Secondly, due to Tabel 6.2 and above explanation, the population as 20 and the generation as 30 were chosen. In summary, the parameters of GA are crossover rate = 0.6; mutation rate = 0.1; population = 20; generation = 30.

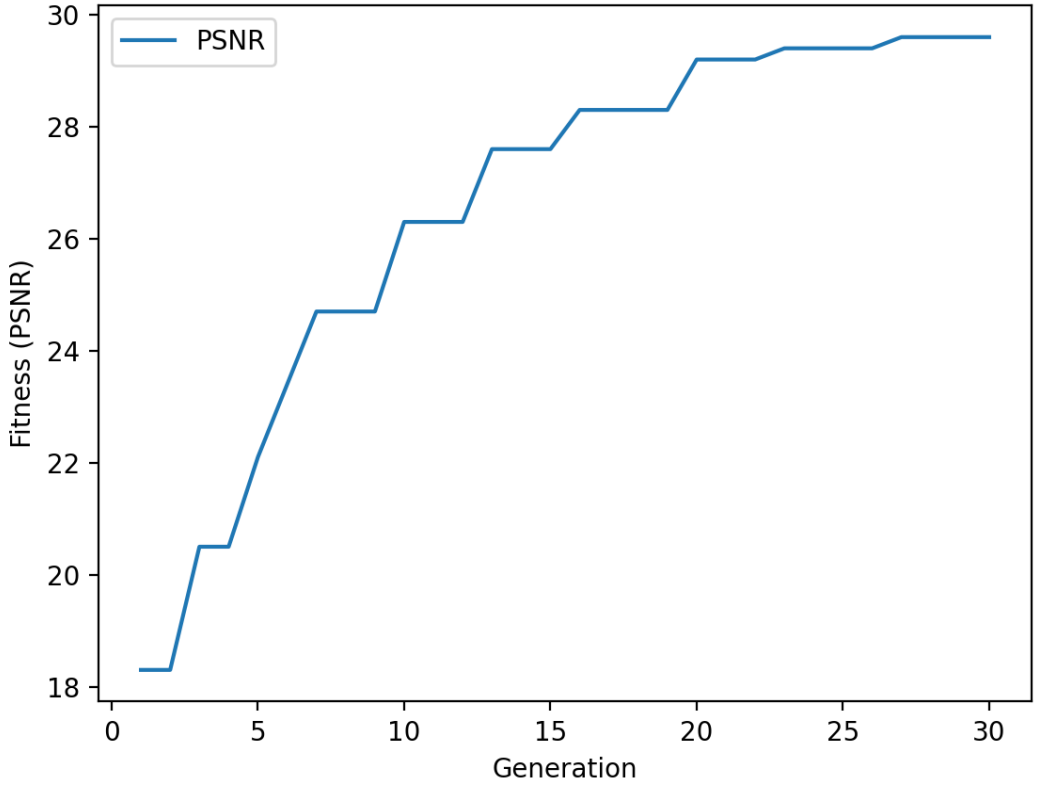

Fig 6.3: Convergence of the best individual fitness with respect to the generation number. (image)

In Figure 6.3, here fitness is defined by PSNR. x-axis is generation and y-axis is fitness. we can see from the graph that as the genetic algorithm is trained fitness gets larger, which means that better and better individuals are selected. At about 25 generations, we can see that the selection of the best individuals has basically stopped. This means that there is no more training value to increase the generation.

6.3.3 Experimental Result

Before optimization on the model, the experiment was performed to compare the number of input images' with the generated images' number. At this stage, the best combination was 8-2. After the optimization by GA, the best combination was found to be 8-1, and the active function set was ReLU-tanh-ReLU-tanh (Table 6.3).

And comparison of the generated image is shown in Figure 6.4.

Table 6.3: Comparison between Original and Optimized

Algorithm	No. hidden layers	Activation function	PSNR	Input-Output
Original	4	ReLU,ReLU,ReLU,tanh	29.1	8-2
Optimized	4	ReLU,tanh,ReLU,tanh	29.6	8-1

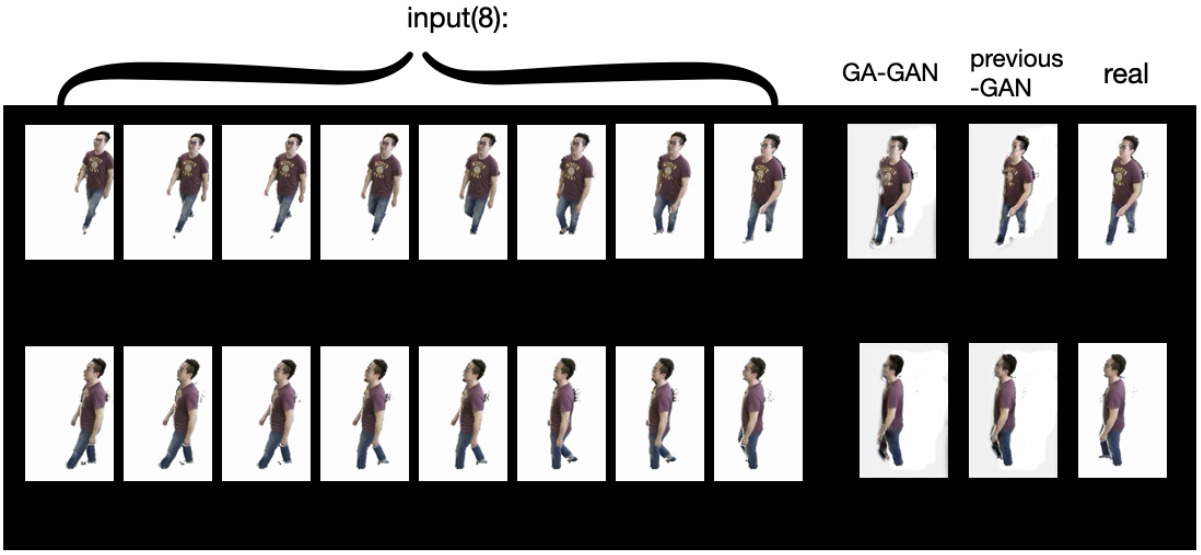

Fig 6.4: Generated image comparison

6.4 Summary

As Chapter 5 showed, this research used a Genetic Algorithm to automatically optimize some hyper-parameters. When using successive-GANs to generate the next frame of a pedestrian image, many hyper-parameters need to be set manually, including the construction of the GANs model. The manual design approach increases the threshold for the design of the model, and it is dependent on the experience and skills of the model designer. The advantage of using a genetic algorithm to optimize GANs model is that hyper-parameters can be set automatically. In this study, the number of input images and output images and the activation function in the Generator were used as the chromosomes of the genetic algorithm for optimization. The results show that the method proposed in this study can improve the quality of the generated pictures. In the previous study, the activation function was always set by ReLU, but in this research, the best combination of activation function is ReLU-Tanh-ReLU-Tanh. This is different from the way traditional GANs are designed. In addition, this research also produced an automated machine learning solution for the analysis and pre-

diction of future time-series image data.

Chapter 7

Conclusion

Machine learning represented by CNN has achieved great success. However, CNN is not very good at data generation and fitting data distribution. Furthermore, the idea to use GANs provides an excellent framework for this field. At present, machine learning is divided into four aspects: supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning. CNN has shown powerful performance in the field of supervised learning and GANs has taken the field of unsupervised learning a big step forward. GANs are composed of generators and discriminators. The advantages of this framework are as follows:

- The model only uses back-propagation, without the Markov chain
- In theory, if a function is differentiable, it can be used to construct Generator and Discriminator, because it can be combined with a deep neural network to make a deep generative model
- The Generator's weights update is not directly from the data sample but uses the back-propagation from the Discriminator (this is also the most different from the traditional method)

Such a framework allows GANs to confront many problems. One of this thesis's most essential research aspects was the fitting distribution and prediction of time-series data. The most crucial point in building a model that can analyze time-series data based on GANs is how to convert the unsupervised learning of the original GANs into semi-supervised learning. After the model was built, the time-series GANs were also a fixed-mode model that had

not been optimized. Currently, the optimization of the model has become essential. Due to the particularity of the time-series data, different time-series data requires models with different hyper-parameters.

This research proposed solutions for the model construction and optimization of GANs when using time-series data. Previous analysis and predictions for time-series include ARIMA, ARCH, RNN, LSTM, and the Markov chain, for example. In this research, the results indicated that GANs showed better performance in fitting and predicting time series data than the previous approaches. This better performance means that GANs have good flexibility and research value in time-series data prediction problems. Secondly, in machine learning represented by CNN, the optimization problem has been studied by many researchers, but in the field of GANs, there have not been many approaches for good optimization, which is also an important topic. This thesis made the following main contributions:

- Time-series (stock numerical data, pedestrian image data), models based on GANs were constructed. These models can be applied not only to stock data but also to the analysis and prediction of other time-series data, such as the walking behavior of pedestrians.

- Through tuning the original GANs (unsupervised learning) into sequential GANs (semi-supervised learning), the sequential GANs have better results than previous single sequential models (ARIMA,LSTM).

- This research uses GA's selection, crossover, mutation, and elimination of natural selection to update essential parameters in the model. Different data sets are used to optimize GANs to different degrees. To get the best performance of prediction, the experimental results show that GA can make sequential GANs perform better and can search the optimal combination of Generator structure and the number of input data.

- Through tuning the hyper-parameters by GA, it was found that different hyper-parameter should be set toward different datasets to get the best results.

Future work: Although this research has achieved good results, using GA to update the hyper-parameters of GANs is still a relatively time-consuming calculation. I hope that in

future research, I can find methods of optimizing GANs that can save calculation time and have good performance at the same time.

Bibliography

[1] J. McCarthy, M. Minsky, and N. Rochester, "Artificial intelligence," tech. rep., Research Laboratory of Electronics (RLE) at the Massachusetts Institute of Technology (MIT)., 1959.

[2] J. McCarthy, M. L. Minsky, N. Rochester, and C. E. Shannon, "A proposal for the dartmouth summer research project on artificial intelligence, august 31, 1955," *AI Magazine*, vol. 27, no. 4, p. 12, 2006.

[3] H. A. Simon, *The shape of automation for men and management*, vol. 13. Harper & Row New York, 1965.

[4] M. L. Minsky, *Computation*. Prentice-Hall Englewood Cliffs, 1967.

[5] S. R. Council, *Artificial Intelligence; a Paper Symposium*. Science Research Council (Great Britain), 1973.

[6] D. Crevier, *AI: the tumultuous history of the search for artificial intelligence*. Basic Books, Inc., 1993.

[7] F.-h. Hsu, "IBM's deep blue chess grandmaster chips," *IEEE Micro*, vol. 19, no. 2, pp. 70–81, 1999.

[8] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European Conference on Computer Vision*, pp. 818–833, Springer, 2014.

[9] A. Dosovitskiy, J. Tobias Springenberg, and T. Brox, "Learning to generate chairs with convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1538–1546, 2015.

[10] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[11] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, "Large-scale video classification with convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1725–1732, 2014.

[12] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3 ed., 2010.

[13] G. E. Hinton and T. J. Sejnowski, *Unsupervised learning: foundations of neural computation*. MIT Press, 1999.

[14] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," *Advances in Neural Information Processing Systems*, vol. 27, 2014.

[15] X. Zhu and A. B. Goldberg, "Introduction to semi-supervised learning," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 3, no. 1, pp. 1–130, 2009.

[16] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of Artificial Intelligence Research*, vol. 4, pp. 237–285, 1996.

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in Neural Information Processing Systems*, vol. 25, pp. 1097–1105, 2012.

[18] S. R. Eddy, "What is a hidden Markov model?," *Nature Biotechnology*, vol. 22, no. 10, pp. 1315–1316, 2004.

[19] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *arXiv preprint arXiv:1411.1784*, 2014.

[20] X. Mao, Q. Li, H. Xie, R. Y. Lau, Z. Wang, and S. Paul Smolley, "Least squares generative adversarial networks," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2794–2802, 2017.

[21] X. Huang, Y. Li, O. Poursaeed, J. E. Hopcroft, and S. J. Belongie, “Stacked Generative Adversarial Networks.,” in *CVPR*, vol. 2, p. 3, 2017.

[22] A. Radford, L. Metz, and S. Chintala, “Unsupervised representation learning with deep convolutional generative adversarial networks,” *arXiv preprint arXiv:1511.06434*, 2015.

[23] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, “Unpaired image-to-image translation using cycle-consistent adversarial networks,” in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2223–2232, 2017.

[24] K. Bousmalis, N. Silberman, D. Dohan, D. Erhan, and D. Krishnan, “Unsupervised pixel-level domain adaptation with generative adversarial networks,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3722–3731, 2017.

[25] M. Molano-Mazon, A. Onken, E. Piasini, and S. Panzeri, “Synthesizing realistic neural population activity patterns using generative adversarial networks,” *arXiv preprint arXiv:1803.00338*, 2018.

[26] T. Schlegl, P. Seeböck, S. M. Waldstein, G. Langs, and U. Schmidt-Erfurth, “f-AnoGAN: Fast unsupervised anomaly detection with generative adversarial networks,” *Medical Image Analysis*, vol. 54, pp. 30–44, 2019.

[27] J. Gui, Z. Sun, Y. Wen, D. Tao, and J. Ye, “A review on generative adversarial networks: Algorithms, theory, and applications,” *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[28] S. Reed, Z. Akata, X. Yan, L. Logeswaran, B. Schiele, and H. Lee, “Generative adversarial text to image synthesis,” in *International Conference on Machine Learning*, pp. 1060–1069, PMLR, 2016.

[29] 川西康友, 出口大輔, 井手一郎, and 村瀬洋, “姿勢を表現する多様体に基づくGANsを用いた物体姿勢推定の検討,” 電子情報通信学会技術研究報告;信学技報, vol. 117, no. 238, pp. 139–144, 2017.

[30] A. Brock, J. Donahue, and K. Simonyan, "Large scale GAN training for high fidelity natural image synthesis," *arXiv preprint arXiv:1809.11096*, 2018.

[31] O. Mogren, "C-rnn-gan: Continuous recurrent neural networks with adversarial training," *arXiv preprint arXiv:1611.09904*, 2016.

[32] C. Esteban, S. L. Hyland, and G. Rätsch, "Real-valued (medical) time series generation with recurrent conditional GANs," *arXiv preprint arXiv:1706.02633*, 2017.

[33] A. Koochali, A. Dengel, and S. Ahmed, "If you like it, GAN it. Probabilistic Multivariate Times Series Forecast With GAN," *CoRR*, vol. abs/2005.01181, 2020.

[34] Y. Li, Z. Gan, Y. Shen, J. Liu, Y. Cheng, Y. Wu, L. Carin, D. Carlson, and J. Gao, "Storygan: A sequential conditional GAN for story visualization," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6329–6338, 2019.

[35] Y. Cheng, Z. Gan, Y. Li, J. Liu, and J. Gao, "Sequential Attention GAN for Interactive Image Editing," in *Proceedings of the 28th ACM International Conference on Multimedia*, MM '20, (New York, NY, USA), p. 4383–4391, Association for Computing Machinery, 2020.

[36] Q. Hoang, T. D. Nguyen, T. Le, and D. Phung, "MGAN: Training generative adversarial nets with multiple generators," in *International Conference on Learning Representations*, 2018.

[37] A. Kendall, Y. Gal, and R. Cipolla, "Multi-task learning using uncertainty to weigh losses for scene geometry and semantics," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 7482–7491, 2018.

[38] O. Sener and V. Koltun, "Multi-task learning as multi-objective optimization," *arXiv preprint arXiv:1810.04650*, 2018.

[39] T. Huster, J. E. Cohen, Z. Lin, K. Chan, C. Kamhoua, N. Leslie, C.-Y. J. Chiang, and V. Sekar, "Pareto gan: Extending the representational power of GANs to heavy-tailed distributions," *arXiv preprint arXiv:2101.09113*, 2021.

[40] X. Lin, H. Chen, C. Pei, F. Sun, X. Xiao, H. Sun, Y. Zhang, W. Ou, and P. Jiang, "A pareto-efficient algorithm for multiple objective optimization in e-commerce recommendation," in *Proceedings of the 13th ACM Conference on Recommender Systems*, pp. 20–28, 2019.

[41] U. Garciarena, A. Mendiburu, and R. Santana, "Analysis of the transferability and robustness of GANs evolved for Pareto set approximations," *Neural Networks*, vol. 132, pp. 281–296, 2020.

[42] B.-Y. Hsueh, W. Li, and I.-C. Wu, "Stochastic gradient descent with hyperbolic-tangent decay on classification," in *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 435–442, IEEE, 2019.

[43] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II," in *International Conference on Parallel Problem Solving from Nature*, pp. 849–858, Springer, 2000.

[44] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[45] K. Deb, "Multi-objective optimisation using evolutionary algorithms: an introduction," in *Multi-objective Evolutionary Optimisation for Product Design and Manufacturing*, pp. 3–34, Springer, 2011.

[46] X. Zan, Z. Wu, C. Guo, and Z. Yu, "A Pareto-based genetic algorithm for multi-objective scheduling of automated manufacturing systems," *Advances in Mechanical Engineering*, vol. 12, no. 1, p. 1687814019885294, 2020.

[47] M. Arjovsky, S. Chintala, and L. Bottou, "Wasserstein generative adversarial networks," in *International Conference on Machine Learning*, pp. 214–223, PMLR, 2017.

[48] I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, "Improved training of wasserstein gans," *arXiv preprint arXiv:1704.00028*, 2017.

[49] X. Wei, B. Gong, Z. Liu, W. Lu, and L. Wang, "Improving the improved training of Wasserstein GANs: A consistency term and its dual effect," *arXiv preprint arXiv:1803.01541*, 2018.

[50] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, "Improved techniques for training GANs," *Advances in Neural Information Processing Systems*, vol. 29, pp. 2234–2242, 2016.

[51] T. Miyato, T. Kataoka, M. Koyama, and Y. Yoshida, "Spectral normalization for generative adversarial networks," *arXiv preprint arXiv:1802.05957*, 2018.

[52] T. Karras, T. Aila, S. Laine, and J. Lehtinen, "Progressive growing of GANs for improved quality, stability, and variation," *arXiv preprint arXiv:1710.10196*, 2017.

[53] K. Roth, A. Lucchi, S. Nowozin, and T. Hofmann, "Stabilizing Training of Generative Adversarial Networks through Regularization," *CoRR*, vol. abs/1705.09367, 2017.

[54] K. Shmelkov, C. Schmid, and K. Alahari, "How good is my GAN?," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 213–229, 2018.

[55] L. Mescheder, A. Geiger, and S. Nowozin, "Which training methods for GANs do actually converge?," in *International Conference on Machine Learning*, pp. 3481–3490, PMLR, 2018.

[56] G. E. Box and D. A. Pierce, "Distribution of residual autocorrelations in autoregressive-integrated moving average time series models," *Journal of the American Statistical Association*, vol. 65, no. 332, pp. 1509–1526, 1970.

[57] T. C. Mills and T. C. Mills, *Time series techniques for economists*. Cambridge University Press, 1990.

[58] G. P. Zhang, "Time series forecasting using a hybrid ARIMA and neural network model," *Neurocomputing*, vol. 50, pp. 159–175, 2003.

[59] T. Bollerslev, "Generalized autoregressive conditional heteroskedasticity," *Journal of Econometrics*, vol. 31, no. 3, pp. 307–327, 1986.

[60] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[61] J. L. Elman, "Finding structure in time," *Cognitive Science*, vol. 14, no. 2, pp. 179–211, 1990.

[62] D. M. Nelson, A. C. Pereira, and R. A. de Oliveira, "Stock market's price movement prediction with LSTM neural networks," in *2017 International Joint Conference on Neural Networks (IJCNN)*, pp. 1419–1426, IEEE, 2017.

[63] T. Fischer and C. Krauss, "Deep learning with long short-term memory networks for financial market predictions," *European Journal of Operational Research*, vol. 270, no. 2, pp. 654–669, 2018.

[64] W. Bao, J. Yue, and Y. Rao, "A deep learning framework for financial time series using stacked autoencoders and long-short term memory," *PloS one*, vol. 12, no. 7, p. e0180944, 2017.

[65] S. Selvin, R. Vinayakumar, E. Gopalakrishnan, V. K. Menon, and K. Soman, "Stock price prediction using LSTM, RNN and CNN-sliding window model," in *2017 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pp. 1643–1647, IEEE, 2017.

[66] S. Siami-Namini and A. S. Namin, "Forecasting economics and financial time series: ARIMA vs. LSTM," *arXiv preprint arXiv:1803.06386*, 2018.

[67] H. Li, Y. Shen, and Y. Zhu, "Stock price prediction using attention-based multi-input LSTM," in *Asian Conference on Machine Learning*, pp. 454–469, PMLR, 2018.

[68] Y. Li, L. Li, X. Zhao, T. Ma, Y. Zou, and M. Chen, "An attention-based LSTM model for stock price trend prediction using limit order books," in *Journal of Physics: Conference Series*, vol. 1575.1, p. 012124, IOP Publishing, 2020.

[69] H. Chung and K.-s. Shin, "Genetic algorithm-optimized long short-term memory network for stock market prediction," *Sustainability*, vol. 10, no. 10, p. 3765, 2018.

[70] S. Takahashi, Y. Chen, and K. Tanaka-Ishii, "Modeling financial time-series with generative adversarial networks," *Physica A: Statistical Mechanics and its Applications*, vol. 527, p. 121261, 2019.

[71] X. Zhou, Z. Pan, G. Hu, S. Tang, and C. Zhao, "Stock market prediction on high-frequency data using generative adversarial nets," *Mathematical Problems in Engineering*, vol. 2018, 2018.

[72] K. Zhang, G. Zhong, J. Dong, S. Wang, and Y. Wang, "Stock market prediction based on generative adversarial network," *Procedia Computer Science*, vol. 147, pp. 400–406, 2019.

[73] T. Kimoto, K. Asakawa, M. Yoda, and M. Takeoka, "Stock market prediction system with modular neural networks," in *1990 IJCNN International Joint Conference on Neural Networks*, pp. 1–6, IEEE, 1990.

[74] E. Guresen, G. Kayakutlu, and T. U. Daim, "Using artificial neural network models in stock market index prediction," *Expert Systems with Applications*, vol. 38, no. 8, pp. 10389–10397, 2011.

[75] X. Ding, Y. Zhang, T. Liu, and J. Duan, "Deep learning for event-driven stock prediction," in *Twenty-fourth International Joint Conference on Artificial Intelligence*, 2015.

[76] A. Borovykh, S. Bohte, and C. W. Oosterlee, "Conditional time series forecasting with convolutional neural networks," *arXiv preprint arXiv:1703.04691*, 2017.

[77] R. Ramezanian, A. Peymanfar, and S. B. Ebrahimi, "An integrated framework of genetic network programming and multi-layer perceptron neural network for prediction of daily stock return: An application in Tehran stock exchange market," *Applied Soft Computing*, vol. 82, p. 105551, 2019.

[78] H. Chung and K.-s. Shin, "Genetic algorithm-optimized multi-channel convolutional neural network for stock market prediction," *Neural Computing and Applications*, vol. 32, no. 12, pp. 7897–7914, 2020.

[79] M. Ranzato, A. Szlam, J. Bruna, M. Mathieu, R. Collobert, and S. Chopra, "Video (language) modeling: a baseline for generative models of natural videos," *arXiv preprint arXiv:1412.6604*, 2014.

[80] W. Lotter, G. Kreiman, and D. Cox, "Deep predictive coding networks for video prediction and unsupervised learning," *arXiv preprint arXiv:1605.08104*, 2016.

[81] Y. Yang, J. Zhou, J. Ai, Y. Bin, A. Hanjalic, H. T. Shen, and Y. Ji, "Video captioning by adversarial LSTM," *IEEE Transactions on Image Processing*, vol. 27, no. 11, pp. 5600–5611, 2018.

[82] Y. Chen, M. Liu, S.-Y. Liu, J. Miller, and J. P. How, "Predictive modeling of pedestrian motion patterns with Bayesian nonparametrics," in *AIAA Guidance, Navigation, and Control Conference*, p. 1861, 2016.

[83] N. Schneider and D. M. Gavrila, "Pedestrian path prediction with recursive Bayesian filters: A comparative study," in *German Conference on Pattern Recognition*, pp. 174–183, Springer, 2013.

[84] S. Bonnin, T. H. Weisswange, F. Kummert, and J. Schmuedderich, "General behavior prediction by a combination of scenario-specific models," *IEEE transactions on Intelligent Transportation Systems*, vol. 15, no. 4, pp. 1478–1488, 2014.

[85] E. Frazzoli, M. A. Dahleh, and E. Feron, "Real-time motion planning for agile autonomous vehicles," *Journal of Guidance, Control, and Dynamics*, vol. 25, no. 1, pp. 116–129, 2002.

[86] 杉本達哉, 坊農真弓, and 佐藤真一, "Rgb-d データによる人物姿勢推定のインタラクション研究への応用," 第 *77*回全国大会講演論文集, vol. 2015, no. 1, pp. 227–228, 2015.

[87] B. D. Ziebart, N. Ratliff, G. Gallagher, C. Mertz, K. Peterson, J. A. Bagnell, M. Hebert, A. K. Dey, and S. Srinivasa, "Planning-based prediction for pedestrians," in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3931–3936, IEEE, 2009.

[88] M. Mathieu, C. Couprie, and Y. LeCun, "Deep multi-scale video prediction beyond mean square error," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2016.

[89] C. Vondrick, H. Pirsiavash, and A. Torralba, "Generating videos with scene dynamics," *Advances in Neural Information Processing Systems*, vol. 29, pp. 613–621, 2016.

[90] Y. Pan, Z. Qiu, T. Yao, H. Li, and T. Mei, "To create what you tell: Generating videos from captions," in *Proceedings of the 25th ACM International Conference on Multimedia*, pp. 1789–1798, 2017.

[91] L. Yu, W. Zhang, J. Wang, and Y. Yu, "SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient.," *CoRR*, vol. abs/1609.05473, 2016.

[92] J. Yoon, D. Jarrett, and M. van der Schaar, "Time-series Generative Adversarial Networks," in *Advances in Neural Information Processing Systems* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), vol. 32, Curran Associates, Inc., 2019.

[93] F. I. Alarsan and M. Younes, "Best Selection of Generative Adversarial Networks Hyper-Parameters using Genetic Algorithm," *SN Computer Science*, vol. 2, no. 4, pp. 1–14, 2021.

[94] U. Garciarena, R. Santana, and A. Mendiburu, "Evolved GANs for generating Pareto set approximations," in *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 434–441, 2018.

[95] X. Feng, J. Zhao, and E. Kita, "Genetic Algorithm-based Optimization of Deep Neural Network Ensemble," *The Review of Socionetwork Strategies*, vol. 15, no. 1, pp. 27–47, 2021.

[96] X. Feng, R. Li, J. Zhao, and E. Kita, "Genetic algorithm based optimization of deep convolutional neural network ensemble for pedestrian moving direction recognition.," *Aust. J. Intell. Inf. Process. Syst.*, vol. 16, no. 1, pp. 56–64, 2019.

[97] L. Xie and A. Yuille, "Genetic cnn," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1379–1388, 2017.

[98] Y. Sun, B. Xue, M. Zhang, G. G. Yen, and J. Lv, "Automatically designing CNN architectures using the genetic algorithm for image classification," *IEEE Transactions on Cybernetics*, vol. 50, no. 9, pp. 3840–3854, 2020.

[99] C. Wang, C. Xu, X. Yao, and D. Tao, "Evolutionary generative adversarial networks," *IEEE Transactions on Evolutionary Computation*, vol. 23, no. 6, pp. 921–934, 2019.

[100] H. Chung and K.-s. Shin, "Genetic algorithm-optimized long short-term memory network for stock market prediction," *Sustainability*, vol. 10, no. 10, p. 3765, 2018.

[101] S. Mitra and S. K. Pal, "Fuzzy multi-layer perceptron, inferencing and rule generation," *IEEE Transactions on Neural Networks*, vol. 6, no. 1, pp. 51–63, 1995.

[102] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[103] D. H. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *The Journal of Physiology*, vol. 160, no. 1, pp. 106–154, 1962.

[104] K. Fukushima and S. Miyake, "Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition," in *Competition and Cooperation in Neural Nets*, pp. 267–285, Springer, 1982.

[105] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[106] E. Fan, "Extended tanh-function method and its applications to nonlinear equations," *Physics Letters A*, vol. 277, no. 4-5, pp. 212–218, 2000.

[107] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," *arXiv preprint arXiv:1505.00853*, 2015.

[108] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3431–3440, 2015.

[109] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," *arXiv preprint arXiv:1412.3555*, 2014.

[110] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[111] D. Whitley, "A genetic algorithm tutorial," *Statistics and Computing*, vol. 4, no. 2, pp. 65–85, 1994.

[112] B. He and E. Kita, "Stock price prediction by using hybrid sequential generative adversarial networks," in *2020 International Conference on Data Mining Workshops (ICDMW)*, pp. 341–347, IEEE, 2020.

[113] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, "PyTorch: an imperative style, high-performance deep learning library," *Advances in Neural Information Processing Systems*, vol. 32, pp. 8026–8037, 2019.

[114] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in PyTorch," in *NIPS 2017 Workshop on Autodiff*, 2017.

[115] Z. Zhang, "Microsoft Kinect sensor and its effect," *IEEE Multimedia*, vol. 19, no. 2, pp. 4–10, 2012.

[116] B. He and E. Kita, "Pedestrian walking direction prediction using generative adversarial networks," in *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 4358–4364, IEEE, 2020.

[117] Q. Huynh-Thu and M. Ghanbari, "Scope of validity of PSNR in image/video quality assessment," *Electronics Letters*, vol. 44, no. 13, pp. 800–801, 2008.

[118] J. Revaud, P. Weinzaepfel, Z. Harchaoui, and C. Schmid, "Epicflow: Edge-preserving interpolation of correspondences for optical flow," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1164–1172, 2015.

Acknowledgments

I would like to express my special gratitude to my supervisor Professor Eisuke Kita. His guidance was indispensable for this doctoral thesis and each academic and journal paper that helped me a lot. He has been a tremendous help to me in conducting my research and writing my dissertation. He helped me make significant growth from zero to one in academics.

Secondly, I would like to thank Professor Takaya Arita and Professor Masahiro Ohka for their insightful and valuable comments on the revision of my dissertation. I would also like to thank all the teachers who taught me and helped me at Nagoya University and Zhejiang Gongshang University. I am very grateful to Nagoya University for all the financial support during my doctoral studies, which allowed me to complete my research work smoothly.

I am grateful to all the people I met in the KITA laboratory, and I have spent some happy times with many of them. I am incredibly thankful to Dr. Feng Xuanang, who has given me lots of guidance in life and academics. He is an outstanding, intelligent and responsible man. I hope he will always be happy in the future.

Finally, I would like to thank my parents, who gave me financial and spiritual support. I feel fortunate and blessed to be their son. I am also very grateful to my girlfriend, Dr. GUO Ling, who gave me a lot of valuable help and advice. I love her very much.