Doctoral Dissertation

Directional Generative Networks: Research on structured object generation methods that do not use pre-prepared datasets

Yasuaki Ito

Supervisor NGUYEN Le Minh

Graduate School of Advanced Science and Technology
Japan Advanced Institute of Science and Technology
Information Science

December, 2023

Abstract

This research proposes a new method of machine learning for structured artifacts. The technique is called Directional Generative Networks (DGN). This study demonstrates that this method has the potential to be industrially applicable. Here, structured artifacts mainly mean industrial products that are physically tangible. Examples are the structure of a building, the molecular structure of a medicine, etc.

Although various industrially applicable machine learning methods have been proposed, their application to the design or search process of physical products such as the above is limited. The reasons for the limited application are: 1) product data and business processes are not easily disclosed as trade secrets, and even if they are disclosed, the amount of data is often insufficient; 2) it is challenging to avoid over-fitting due to a lack of data and 3) the difficulty of avoiding biases originating from the data due to the small amount of data. The performance of machine learning depends on the quantity and quality of the data set. It is presumed that companies dealing with products with physical entities must use machine learning sparingly for the above reasons.

In this study, the structure of physically tangible products is focused on. DGN, the proposal method, is examined to compensate for the lack of data by combining a function that evaluates based on the structure with a machine learning model. Methods or functions to evaluate a product based on its structure are often publicly available. One of the neural network models that make up the DGN is an approximation function of the evaluation function. It is well-known that a neural network with multi-layers can be an approximate function for any function. The scheme of DGN, combining the approximate function as an estimator with a generator, is similar to Generative Adversarial Networks(GAN) generator and discriminator. However, the way ground truth is given differs between GAN and DGN. The return value of the evaluation function is used as the ground truth when training the estimator in DGN. When training the generator, the weights of the estimator, the approximate function model, are fixed. The favorable value the designer wants as the evaluation function output is given as a constant target value. This gives the direction of the training to the generative model. In DGN, the generator and the estimator are trained alternately, like the generator and discriminator are trained alternately in GAN. If the estimator is trained enough, the outputs of the estimator are almost the same as the outputs of the evaluation functions. If the generator is trained enough, the outputs of the generator represent desired products that obtain the desired values of evaluation functions' results. Due to the methods above, you do not have to prepare datasets for the training.

Evolutionary Algorithms(EA) use not training data but feedback from the environment. EA using the evaluation functions as the environment can get suitable results. However, if the complexity of the product parts combination is increased, it will become challenging to get the desired results. On the other hand, due to training DGN with a gradient of exploration field, DGN may get more suitable results than EA in complex problems.

In this research, DGN was applied to the structural design of buildings and the molecular discovery of drugs. The appropriate representation of data and model structure are examined for each task and confirmed its usefulness for both tasks. If the generator and the estimator are well-trained, the generator can generate various products with the desired structure with a sound

output(s) of the evaluation function.

Structural design of buildings and molecular search for pharmaceuticals are completely different industrial fields. It was confirmed that the DGN method can be used in these technologically distant industrial fields. Therefore, it may be applied in various industrial fields, even if a sufficient data set cannot be prepared for the intended training.

Keywords— Generative Model, Unsupervised Learning, Evolutionary Algorithms, Generative Adversarial Networks

Acknowledgements

As a part-time student, my research activities had progressed slowly. It was taken over a long period, about six years. Due to hoping to apply deep learning to a wide range of industries, this research has taken an unusual approach that relies less on data. This did not immediately lead to the desired results, and I frequently lost my way. Despite this, Professor Nguyen had encouraged me calmly. I would like to express my sincere gratitude to my advisor, Professor Nguyen, for his patience and enthusiastic guidance. Without him, I would not have been able to complete this research.

Finally, I would like to thank my wife, Tomoko, and my sons, Sota and Yuta, for understanding and supporting my research.

Contents

1 Introduction **1**
- 1.1 Introductory remarks . . . 1
- 1.2 Machine learning in industrial fields . . . 2
  - 1.2.1 Structures in products . . . 2
- 1.3 Proposed method . . . 3
- 1.4 Research objective . . . 3
- 1.5 Dissertation outline . . . 4

2 Related Works **5**
- 2.1 Evolutionary Algorithms and Genetic Algorithms . . . 5
  - 2.1.1 GA for Structural Design . . . 5
  - 2.1.2 GA for Molecular Search . . . 6
- 2.2 Applications with Deep Learning . . . 6
  - 2.2.1 Structural design . . . 6
  - 2.2.2 Molecular Search . . . 6
- 2.3 Reinforced Learning . . . 8

3 Directional Generative Networks **10**
- 3.1 Elemental methods of DGN derived from deep learning . . . 10
  - 3.1.1 Artificial Neural Networks as Approximate Functions . . . 10
  - 3.1.2 Methods related to deep learning . . . 11
  - 3.1.3 Generative Adversarial Networks (GAN) . . . 14
- 3.2 Directional Generative Networks(DGN) . . . 15
  - 3.2.1 Structure of DGN . . . 15
  - 3.2.2 Comparison between GAN and DGN . . . 15
  - 3.2.3 Theoretical description . . . 17
- 3.3 Simple Examples of DGN . . . 20

4 DGN Application for Structural Design **25**
- 4.1 Structural Design . . . 25
  - 4.1.1 Structural Design . . . 25
  - 4.1.2 Domain Knowledge . . . 26
- 4.2 DGN for Structural Design . . . 27
  - 4.2.1 Model Structure . . . 27
  - 4.2.2 Data Representaion . . . 28
  - 4.2.3 Training . . . 28
- 4.3 Experiment . . . 32
  - 4.3.1 Measurments . . . 32
  - 4.3.2 Method . . . 32
  - 4.3.3 Result . . . 33
- 4.4 Summary of the Chapter . . . 34

5 DGN Application for Molecular Search **37**
5.1 Molecular Search . . . 37
5.2 DGN for Molecular Search . . . 38
5.2.1 Model structure . . . 38
5.2.2 Data Representation . . . 45
5.2.3 Training . . . 45
5.3 Experiment . . . 47
5.3.1 Measurements . . . 47
5.3.2 Comparison . . . 47
5.3.3 Result . . . 48
5.3.4 Degree of freedom of DGN input . . . 50
5.4 Summary of the Chapter . . . 57

6 Conclusion **58**
6.1 Findings . . . 58
6.1.1 Applicability of DGN . . . 58
6.1.2 Advantage of DGN over EA . . . 59
6.1.3 Elimination of bias derived from Data . . . 59
6.2 Future works . . . 59

7 Appendix **61**
7.1 Structural Design Method Based on Japanese Standard Act . . . 61
7.2 Libraries and Resources . . . 63
7.3 DGN for molecular finding; model details . . . 64

Bibliography **75**

Publications **78**

List of Figures

2.1 GAN for shear panel Layout . . . 7
2.2 ORGAN . . . 7
2.3 MolGAN . . . 8

3.1 Highly nonlinearity objective function . . . 11
3.2 Residual Networks Block . . . 12
3.3 U-Nets . . . 13
3.4 Visual Self-Attention . . . 14
3.5 GAN diagram . . . 15
3.6 DGN diagram . . . 16
3.7 Training Procedure of DGN . . . 16
3.8 Comparison of training procedure . . . 18
3.9 Loss value transition of estimator model . . . 21
3.10 Loss value transition of combined model . . . 21
3.11 Calculated value y_0 transition . . . 22
3.12 Generated value y_1 transition of DGN . . . 22
3.13 Distribution of generated values for function F_0, using addition and multiply . . . 23
3.14 Distribution of generated values for modulo function . . . 24

4.1 DGN Training Scheme for Structural Design . . . 28
4.2 Building Data Representation Simplified Examples . . . 29
4.3 Data Representation of Generated Shear-Panels . . . 30
4.4 Training Procedure; DGN for Structural Design . . . 31
4.5 Building Data Elements . . . 32
4.6 Estimators' loss values transition during training . . . 33
4.7 Generators' loss values transition during training . . . 33
4.8 Generators' loss values with test data . . . 34
4.9 Eccentricity transition with test data during training . . . 35
4.10 DCR transition with test data during training . . . 36

5.1 DGN Training scheme for Molecular Generation . . . 39
5.2 Generator . . . 40
5.3 Estimator . . . 41
5.4 U-net Block . . . 42
5.5 Self-Attention Block . . . 43
5.6 Residual Block . . . 44
5.7 Data representation example . . . 45
5.8 Training Procedure; DGN for molecular Search . . . 46
5.9 Random Generation Procedure . . . 48
5.10 Evolutionary Algorithm Procedure . . . 49
5.11 DGN Validation Procedure . . . 49
5.12 The kernel density estimates of the score distribution according to the number of molecules 51
5.13 Molecular Examples with Random Generation . . . 52
5.14 Molecular Examples with Evolutionary Algorithm . . . 53

5.15 Generated molecular examples with DGN 54
5.16 QM9 Molecular examples 55
5.17 Comparison of degree of freedom 56
5.18 Comparison to Sum6 score between DGN and DGNr 56

7.1 DGN for molecular finding; generator 0 65
7.2 DGN for molecular finding; generator 1 66
7.3 DGN for molecular finding; generator 2 67
7.4 DGN for molecular finding; generator 3 68
7.5 DGN for molecular finding; generator 4 69
7.6 DGN for molecular finding; estimator 0 70
7.7 DGN for molecular finding; estimator 1 71
7.8 DGN for molecular finding; estimator 2 72
7.9 DGN for molecular finding; estimator 3 73
7.10 DGN for molecular finding; estimator 4 74

List of Tables

3.1 Comparison between GAN, and DGN 19

5.1 Results of DGN and comparison with EA, random generation, and previous studies . . . 50

Chapter 1

Introduction

1.1 Introductory remarks

We live surrounded by many products. Smart devices, clothing, cutlery, and furniture we use daily. In the big ones, houses, cars, bridges, buildings. For the small ones, gears in clocks, ICs, CPUs, tablets, etc., to name but a few. We live every day with the benefits of these products.

Around 2022, generative models originating in deep learning had a significant social impact. Generative models return various products for a single input, for example. For a sentence in the input, the Large Language Model (LLM) generates an appropriate sentence to answer the input. An image generation model would generate an image corresponding to the input sentence. These products are not necessarily uniquely determined but can be generated in various ways, and the user selects the desired solution that meets their needs. Under these circumstances, it is natural to wonder if machine learning methods using techniques such as deep learning could be applied to the business process of product development. It becomes a desirable tool if you can design the product you want to create by entering the desired attributes.

Various machine-learning methods have been proposed for industrial applications. Although various industrially applicable machine learning methods have been proposed, their application to the design or search process of physical products such as the above is limited. The reasons for the limited application are: 1) product data and business processes are not easily disclosed as trade secrets, and even if they are disclosed, the amount of data is often insufficient; 2) it is challenging to avoid over-fitting due to a lack of data and 3) the difficulty of avoiding biases originating from the data due to the small amount of data. The performance of machine learning depends on the quantity and quality of the data set. It is presumed that companies dealing with products with physical entities must use machine learning sparingly for the above reasons.

Naturally, in aiming for industrial use, there are many constraints, such as physical, legal, economic, etc., and there will be a mixture of items for which fluctuations in solutions are acceptable and items for which they are not. It could significantly impact the industry if a generative model can be applied to the design process while respecting these constraints.

This research proposes a new method of machine learning for structured artifacts. The technique is called Directional Generative Networks (DGN). This method can be used even when datasets for training are insufficient. This study demonstrates that this method has the potential to be industrially applicable. Here, structured artifacts mainly mean industrial products that are physically tangible. Examples are the structure of a building, the molecular structure of a medicine, etc.

In this research, DGN is applied to the structural design of buildings and the molecular discovery of drugs. Structural design of buildings and molecular search for pharmaceuticals are completely different industrial fields. If it is confirmed that the DGN method can be used in these technologically distant industrial fields, it may be applied in various industrial areas.

1.2 Machine learning in industrial fields

There are many industrial applications of machine learning. Technologies based on deep-learning methods have made significant progress in recent years. Typical applications include the following;

- image recognition, which forms the basis of automated driving technology
- lesion detection or disease diagnosis in CT or X-ray images
- Image generation using stable diffusion models, etc.
- Sentence generation using large language models.

These applications are supported by large, high-quality data sets as well as various algorithmic innovations. If good-quality data is available enough for training, the desired results can be obtained in the space defined by the dataset. In the case of lesion detection or disease diagnosis, if the results obtained by a trained model are as accurate or more accurate than the judgment of a physician or other expert, it can be a helpful tool. Issues that require social consensus, such as responsibility for the output of the model for industrial use, are separate from the subject of this study.

On the other hand, some industrial fields do not often use machine learning or similar methods despite various suggestions. For example, some generative models are derived from Generative Adversarial Networks(GAN) [3] and Evolutionary Algorithms(EA) [18] for molecular search. GAN-like method relies on datasets for training. In industrial fields, dataset preparation has often become a huge problem. For EA-like methods, the combinatory explosion is the main obstacle to real problems. Real-world problems are often too complex to apply EA methods.

Commercial companies are reluctant to disclose experimental data obtained in research and development or product development except for strategic purposes. Therefore, those data are rarely disclosed to the other companies. In addition, the data collected at each company may be in a unique form at the discretion of each person in charge unless the data formats are aligned with a corporate strategy. They may need to be more neatly organized for sharing to use for machine learning. Even if a company strategically collects data by aligning data formats, there is a limit to the amount of R&D budget that a single company can invest, and there is a limit to the amount of data that can be collected. For these reasons, the amount of data that can be used for machine learning is often insufficient.

When data is scarce, predictive models may overfit the insufficient data and fail to achieve sufficient generalization ability. Even if they can be trained with poor data to some extent, they may be strongly influenced by the biases of the data set. Therefore, reducing the dependence of machine learning models on the datasets would be a great advantage for their industrial use.

One method that does not use a pre-prepared data set for training is the evolutionary algorithm (EA). Instead of datasets, it receives feedback from the environment and searches for appropriate solutions. EA performs very well for some problems. However, for complex problems or problems with many combinations of genes that make up the solution, EAs are known to fail to provide empirically reasonable solutions.

1.2.1 Structures in products

To cope with these issues, this research focused on the structure of products. Here, products mean tangible objects such as building structures, molecule structures in medicine, etc. Almost without exception, tangible products have a specific structure and were designed by someone. The designers determine a more suitable structure for each product through the design processes. They were conceived to make someone's life better through the products and were shaped in the pursuit of economic rationality.

Regarding building structures, wood, steel, and reinforced concrete structures are typical structural types. These structures have columns, beams, and floor slabs, and shear-bearing walls are provided as needed. The required space is defined according to the use of the building, and the structural members are determined so that the room can be maintained. The required space is an artificial requirement or constraint in structural design, while the ability to resist gravity, seismic loads, and wind loads is primarily a physical constraint. Other typical constraints include technical standards, building codes, and economics.

In drug development, molecules that are effective for a particular disease or condition are searched for. Its efficacy is derived from the structure of the molecule. The main physical constraints are the bonds between the atoms that make up the molecule. Other constraints are whether the molecule is toxic, stable under normal temperature and pressure conditions, and can be synthesized economically and rationally.

These industrial products come from different industrial sectors, and the structure of each product is very different. Due to the measurements above, it can measure the quality or performance derived from its structures. In this research, we focus on the fact that evaluation is possible based on structure and propose a new method of applying generative models derived from deep learning.

1.3 Proposed method

Some tasks performed by human designers can be considered functions that return deliverables in response to clients' requests as arguments. It is also known that neural networks can approximate any function. If tasks performed by human designers are considered functions, formulating the function correctly may not be possible. Regarding the human creative act of drawing, we draw a picture as an output based on some request or trigger.

In the case of drawing an image of a cat as an example, even if we show a procedure for arranging the position and color of the cat's parts (eyes, ears, whiskers, legs, etc.) as program code, it is impossible to draw a variety of cat pictures. If we try to deal with various representations, the algorithm becomes more complex, and we will soon face difficulties. If a human prepares the algorithm, they will likely only be able to design a model with poor generalization capability. Therefore, to generate images, it is necessary to use models derived from deep learning, such as GANs. Although models such as GAN can be expressed in meta-equations, it is challenging to explain the meaning of the values of the parameters of each layer in a way that humans can intuitively understand.

The relationship between the inputs and outputs of those generative models is similar to the act of design in the industrial field. The designer responds to the client's ambiguous request (input) with various design alternatives. The clients select the most desirable proposal from the multiple proposals. The generative model can be applied to designing or similar tasks from this similarity.

This research proposes Directional Generative Networks(DGN). DGN aims to compensate for the lack of data by combining a function that evaluates based on the structure with a machine learning model. Methods or functions to evaluate a product based on its structure may be publicly available. In a DGN, one of the neural network models that make up the DGN is an approximation function of that evaluation function. It is well-known that a neural network with multi-layers can be an approximate function for any function. The scheme of DGN, combining the approximate function as an estimator with a generator, is similar to Generative Adversarial Networks(GAN) generator and discriminator. However, the return value of the evaluation function is used as the ground truth when training the estimator in DGN. When training the generator, the weights of the estimator as the approximate function model are fixed, and the desired output value of the evaluation function is given as a constant target value. This gives the direction of the training to the generative model.

In DGN, the generator and the estimator are trained alternately, like the generator and discriminator are trained alternately in GAN. If the estimator is trained enough, the outputs of the estimator are almost the same as the outputs of the evaluation functions. If the generator is trained enough, the outputs of the generator represent desired products that obtain the desired values of evaluation functions' results. Because of the technique, you do not have to have pre-prepared datasets for the training.

1.4 Research objective

This research aims to confirm the applicability of DGN, the proposed method, to real-world problems. The real-world problems are the design/exploration of building structures, molecular structures, etc., objects with structures. Building structural design and molecular exploration are completely different technical fields. By confirming the applicability of the proposed method DGN to such distant domains, we can

verify its generality. Just as the applicability of genetic algorithms has been confirmed in a wide range of fields, we believe that the proposed method can be applied in various areas.

In the structural design of buildings, the feasibility of a structural member arrangement that satisfies some of the requirements of the building code is confirmed. The placement of shear walls, even when intuitively positioned, rarely results in an arrangement that meets the criteria. Generally, calculations are performed on the placed shear walls to see if they meet the criteria, and then they are repositioned based on the values. The process is repeated until the criteria are satisfied. If the model has a reasonable probability that the solution candidates it generates include those that satisfy the criteria, it can be considered to have reached an industrially valuable level.

For molecular search, we aim to obtain a score for molecules obtained by the proposed model DGN that is comparable to prior methods. The prior methods to be compared are the genetic algorithm and the derivative model of GAN.

1.5 Dissertation outline

This dissertation is organized as follows.

- Chapter 2 describes previous machine learning studies that tried to apply industrial problems. It includes explaining the applications of EA, deep learning, and reinforced learning.
- Chapter 3 explains DGN, the method proposed in this study. This chapter contains the basic concepts of DGN, training procedures, and a simple example of DGN. It includes a comparison of training procedures between GAN and DGN. The calculation process of EA and DGN are also compared.
- Chapter 4 explains DGN application for structural design. The results show that the DGN is at a level that can be used industrially. Most of the generated building structure results, calculated with evaluation functions, satisfied the criteria defined by the Japanese Building Standards Act.
- Chapter 5 explains DGN application for molecular search. It is a DGN example that is close to a real problem. Here are explanations that include the data representation method and the learning method of DGN for molecular search. Here, DGN is compared to the supervised learning methods of previous studies. The comparison shows that DGN's performance is comparable with previous methods. In addition, it has been confirmed that DGN has an advantage over EA for complex problems.
- In Chapter 6 shows findings obtained from this study. The usefulness and limitations of DGN are discussed. Based on the findings, future research issues are listed.
- Chapter 7 is an appendix describing an excerpt of the structural design methodology based on the Japanese Building Standard Act used in Chapter 4. It also presents the computational resources and libraries used in the experiments in Chapter 5 and the details of the DGN model in Chapter 5.

Chapter 2

Related Works

2.1 Evolutionary Algorithms and Genetic Algorithms

Since the first evolutionary algorithm was proposed by J. H. Holland [18] for the first time, it has been applied in various fields. A characteristic feature of evolutionary algorithms is that the survival of the generated individuals is determined based on feedback from the environment. The environment model can remain a black box if feedback can be obtained. This is expected to be a significant reason why applications have been attempted in various fields.

Evolutionary algorithms have the disadvantage of being difficult to apply to complex problems. It is challenging to apply evolutionary algorithms when the number of possible solutions increases exponentially with the size of the problem. Finding a reasonable solution can be challenging in such a vast search space because EA is based on a random search. [5] [44] [38]. To compensate for this, genetic algorithms using search strategies such as crossover and selection were used to improve evolutionary algorithms, making the discovery of suitable solutions more efficient. However, even with genetic algorithms, difficulties due to combinatorial explosion are inevitable.

2.1.1 GA for Structural Design

In high-rise buildings, dampers that reduce vibration amplitude may be installed to control horizontal deformation caused by earthquakes or wind. The designer determines the place of dampers in a building, conducts a response analysis of the structure for the arrangement, and readjusts the placement of the dampers based on the results of the analysis.

Generally, damper locations are determined by the designer's experience. N . Wongprasert et al. attempted to optimize this damper placement using GA [45]. The damper arrangement is improved by evaluating the maximum amplitude of the wave motion as the same as the external force to reduce the amplitude. By doing so, they have obtained a damper arrangement that reduces the response of the building.

S. A. Faizi et al. applied GA to the problem of placing buckling-restrained braces in Reinforced Concrete frames [11]. Buckling-restrained braces and other diagonal members, when incorporated into moment frames, become like walls, which may affect the usability of the building. Stronger braces can resist horizontal forces with fewer braces but cannot be unnecessarily strong because they increase the cost of the surrounding columns and beams. Braces of appropriate strength should be placed where they do not interfere with the floor plan.

The designer's experience will determine the appropriate placement, but this is an example of where an optimization method such as GA can obtain a better arrangement. Both examples are not so complex to solve with GA.

2.1.2 GA for Molecular Search

Within the research area of the searching method for new molecules with desirable features, Lee et al. proposed a method of stacking two GA models [26]. This is intended to focus on a desired region of the search space. The method uses the similarity to a specific molecule prepared from the data set and the desired features as a score for individual selection.

This study is unique in that GA was applied in two stages. As mentioned, in complex problems, EA or GA is difficult to converge to an appropriate solution due to combinatorial explosion. In contrast, by dividing the problem into two stages, GA can be easily applied to simple problems. Thus, GA can be helpful if the problem can be simplified with domain knowledge.

2.2 Applications with Deep Learning

2.2.1 Structural design

Y. Fei et al. [13] attempted to place shear panels on reinforced concrete structures with GAN. The GAN model achieved the same quality design as a highly efficient human expert. The generator input is CAD drawings images without shear panels. It generates shear panel arrangements for the drawings. A discriminator is applied to this generated shear panel drawing to determine whether it is an image with real or fake shear panels. In addition, the relative displacement of each floor as physical performance is fed back to the generator as an indicator to encourage a more favorable generation 2.1.

This method depends on a shear-panel dataset. In general, the placement of shear panels is not determined by a single solution but rather by various options. Therefore, there is a concern that methods that use shear panel data created by humans as the correct answer will limit the variety of solutions that can be obtained. In addition, it often requires significant expense to collect the data.

2.2.2 Molecular Search

ORGAN

ORGAN [16] is combined with conventional GAN and reinforcement learning. Figure 2.2 retrieved from [16] is the schema for ORGAN. Objective-Reinforced Generative Adversarial Networks (ORGAN) combines Generative Adversarial Networks (GAN) with Reinforcement Learning (RL) techniques. This method was applied to protein generation and music generation. Remarkable performance was achieved for these tasks.

The ORGAN model trains a network of generators and discriminators through adversarial training to generate data with various orders, e.g., protein sequences or musical notes in music. As in GAN, the generators generate sequences, which the discriminators then evaluate. The discriminator is true/false classifier trained with real data and synthesized data. The two models can be trained adversaries to generate new ordered data with the characteristics of the original data.

ORGAN combines the GAN method with reinforcement learning techniques. It rewards them for following a particular desired pattern or structure. This reward is used to update the generator's parameters, such as weights. This encourages the generative model to generate more favorable sequences. The parameter λ is used for tuning the weight of feedback between the discriminator and reward.

By combining adversarial training with reinforcement learning, ORGAN has shown promising results in sequence generation tasks such as music composition and protein sequence generation. However, the performance depends on datasets. If the datasets are insufficient to obtain favorable results, this method does not necessarily work well.

MolGAN

Like ORGAN, MolGAN [9] combines GAN and reinforcement learning 2.3. Unlike ORGAN, MolGAN uses Graph Convolutional Networks (GCN) for its model. GCN is excellent at estimating molecular properties. [23]

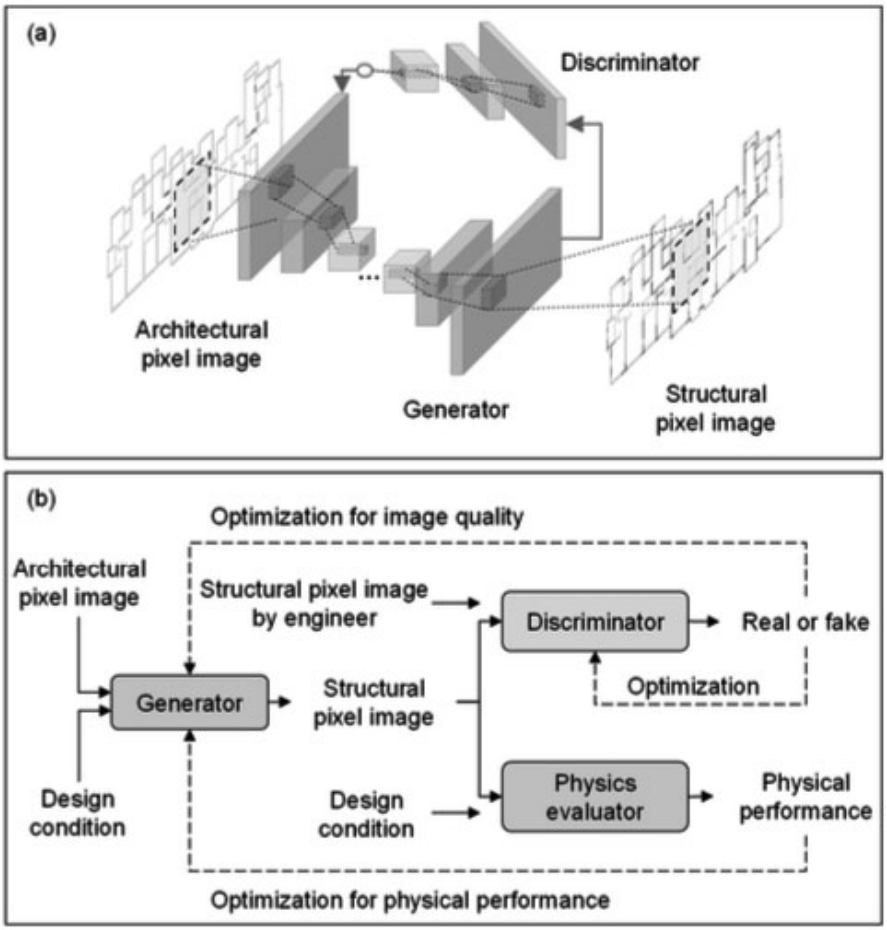


Figure 2.1: GAN for shear panel Layout: image is retrieved from original paper [13]. On top, the generator input is CAD drawing images without shear panels, generating shear panels for the drawing. In the bottom image, the diagram's grey rectangle is a discriminator. It is applied to the generated shear-panel drawing to determine whether it is from real data or generated. The relative displacement of each floor that physical performance is fed back to the generator as an indicator to encourage a more favorable generation.

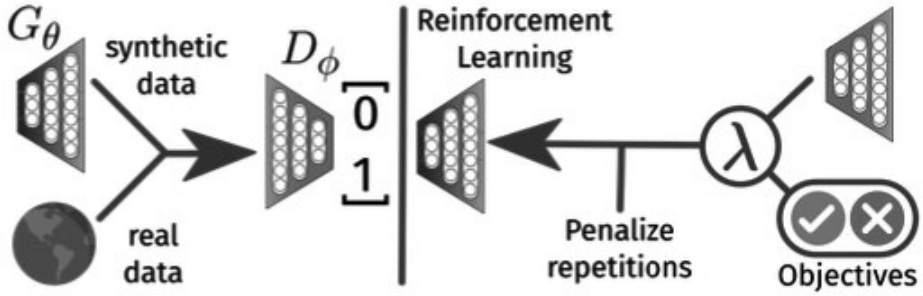


Figure 2.2: ORGAN; GAN for molecule generation: image is retrieved from original paper [16]. ORGAN combines this GAN method with reinforcement learning techniques. It rewards them for following a particular desired pattern or structure. This reward is used to update the generator's parameters, such as weights. This encourages the generative model to generate more favorable sequences. The parameter λ is used for tuning weight of feedback between discriminator and reward.

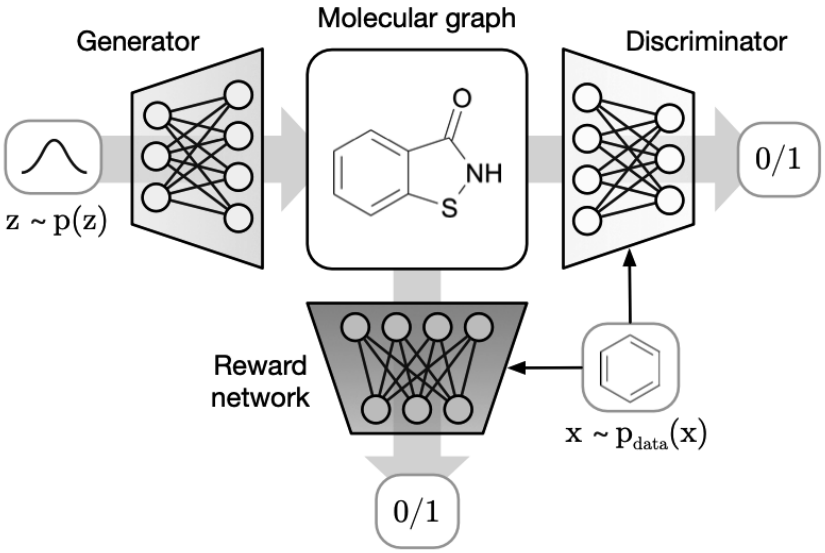


Figure 2.3: MolGAN; GAN for molecule generation: image is retrieved from original paper [9]. MolGAN has a generator, discriminator, and reward network. The models are based on Graph Convolutional Networks (GCN). The green model is generator. The yellow model is discriminator. The blue model is reward network.

MolGAN has a generator, discriminator, and reward network. The models are based on GCN. The generators and discriminators are trained in the same way as in GAN. The discriminator identifies the molecules generated by the generator or the molecules from the dataset.

The molecules generated are rewarded for having desirable features through the reward network. This allows for the generation of molecules with desirable features. Similar to ORGAN, the weight between the discriminator and the reward network is hyperparameter λ.

MolGAN has shown promising results in generating novel and diverse molecular graphs that closely resemble the properties found in real molecules and is expected to have applications in drug discovery and materials science. On the other hand, it is prone to mode collapse and challenging to train. It is not easy to generate molecules with more desirable characteristics because there is a small number of molecules with desirable features in the dataset. In MolGAN experimental results, there is a high probability that the generated molecules are included in the original dataset. Also, the number of generated molecules is small, which is not included in the original dataset and has desirable features. The performance of this method is also inevitably dependent on the datasets.

2.3 Reinforced Learning

Reinforced learning [39] refers to methods by which a model is trained with feedback from the environment. Reinforcement learning is a type of machine learning. An agent learns by acting in an environment and receiving rewards or penalties. The agent aims to maximize the cumulative reward over time or state. The environment is formulated as a Markov decision process.

The learning procedure for reinforcement learning is as follows:

1. the agent acquires the current state of the environment.
2. Based on the current policy, the agent chooses an action to execute.
3. The environment probabilistically transitions to a new state.
4. The agent receives a reward or penalty based on its action.

5. The agent updates its policy based on the observed reward and the new state using a value function estimation technique.

Reinforcement learning algorithms are distinguished by policy type, whether the action space is continuous or discrete, and whether the state space is continuous or discrete. Prominent reinforcement learning algorithms include the SARSA, Q-learning [41] [40], and Deep Q-Networks (DQN). For example, the SARSA method deals with discrete values in action and state space. Q-learning and DQN treat the action space as discrete and the state space as continuous. Both have a Q value that indicates the validity of a rule and updates the value each time the agent acts. DQN is characterized by its Q table being a neural network.

Reinforcement learning has been applied to robot control systems, recommendation systems, and other applications. In recent years, DQN has made headlines by outperforming humans in gameplay [28].

Deep Q-Networks

DQNs were introduced by Mnih et al. [28]. Deep Q-Networks (DQNs) is a reinforcement learning algorithm that uses deep neural networks to approximate the Q function in Q-learning. The Q-function measures the expected cumulative reward for taking a particular action in a given state and following the policy. DQN substitutes a deep neural network for the Q function, obtaining a more accurate and flexible representation than previous methods.

DQN has several innovations to improve performance and stabilize learning.

1. The agents initially act randomly to avoid falling into a local optimum. Gradually, the proportion of random actions is reduced. As a result, the ratio of actions by policy is increased.

2. Clip the error of the loss function. When the absolute value of the error exceeds a threshold value, the error is clipped at that value to prevent divergence during learning.

3. Update the weights of the Target Network at regular intervals.

DQN uses the Replay Buffer and Target Network to stabilize the learning process and prevent agents from overreacting to the environment. The Replay Buffer stores the agent's experience (state, action, reward, next state). The target network is a copy of the main network used to compute target Q values during training. The target network is updated less frequently than the main network, which helps stabilize the training process.

DQN has attracted attention because it outperforms humans in gameplay. It has many applications, including robot control and autonomous driving.

However, these methods are difficult to stabilize training because they aim for future rewards and are trained in time history. Most models for structured product design do not need to be trained in time history. Therefore, it is not necessarily suited for tangible product design.

Chapter 3

Directional Generative Networks

3.1 Elemental methods of DGN derived from deep learning

At the beginning of neural networks, it was known that neural nets could approximate an arbitrary function by layering units of neurons. However, there was a drawback: the deeper layers, the more challenging to train. This problem was overcome by J. Hinton, I. Goodfellow, Y. Bengio, et al. They used a computational graph-based differentiation method called backpropagation to achieve a dramatic performance improvement in image recognition. That is the beginning of deep learning. Subsequently, their application research progressed, and the method was applied not only to image recognition but also to speech recognition, automatic translation, and a wide variety of other applications. In recent years, the Large Language Model (LLM) has been widely recognized, and its use as a sentence generator or search tool has been progressing, significantly impacting society.

This section shows methods related to deep learning used in Directional Generative Networks(DGN).

3.1.1 Artificial Neural Networks as Approximate Functions

Artificial neural networks(ANN) have been proven to approximate arbitrary functions [8] [19]. A nonlinear function is used as the activation function of a neuron. By stacking its units, a function of complex shapes can be represented. Therefore, it can be approximated with arbitrary precision by the number of neurons in the hidden layer of the neural network. This shows that the neural network can be applied to the classification problem by one-hot encoding and the regression problem.

In the case of using a model based on a neural network as a generative model in the industrial or engineering field, one idea is to connect the evaluation function used to evaluate the design proposal with the generative model and update the weights of the generative model based on the error between the evaluation value of the generated one and the desired evaluation value. This is a possible method. However, evaluation functions used in industry or engineering are not necessarily differentiable. In some cases, such as when the formula applied diverges depending on the conditions, the function may become discontinuous at the point of divergence. In such cases, the problem arises that differential methods such as error backpropagation cannot be applied.

Therefore, approximating the evaluation function with a model based on a neural network may solve the problem. The approximation formula will have an error with the underlying function, but if the error is within an acceptable engineering range, it may be an industrially valuable method.

Clipping Norm of Gradients

When a differentiable function approximates a non-differentiable function with a discontinuity or singularity, the model's behavior during training may become unstable near the discontinuity or singularity. When attempting to update the weights by gradient descent, a sudden change in value can cause mode collapse 3.1. To avoid this, gradient clipping may be effective. As noted by I. Goodfellow et al. [14], for example, highly nonlinear functions computed over many time steps tend to have very large or small gradients. In a search space of such a shape, if gradient clipping is not applied, the position being searched

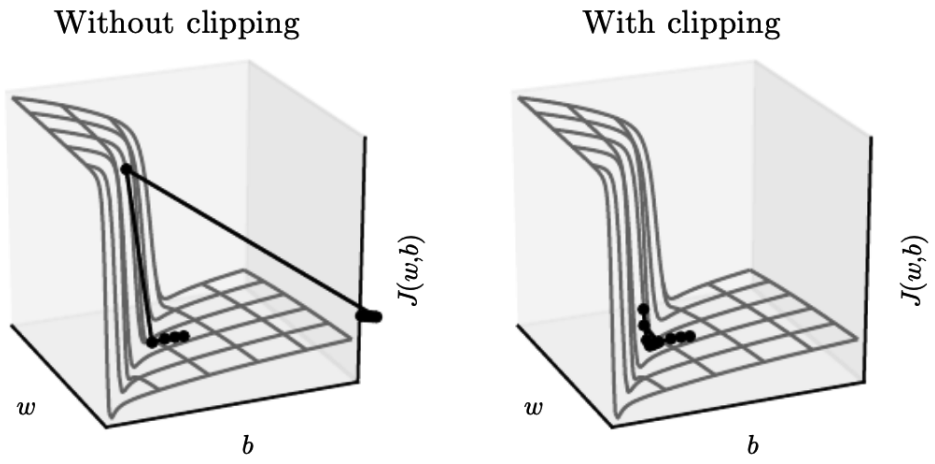


Figure 3.1: Highly nonlinearity objective function: This image is retrieved from the book Deep Learning [14]. Highly nonlinear functions computed over many time steps tend to have very large or small gradients. In a search space of such a shape, if gradient clipping is not applied, the position being searched may be moved significantly when the gradient changes rapidly, that shown in the left-hand image, wasting the optimization process that has been performed. The right-hand image shows the behavior If the clipping is applied.

may be moved significantly when the gradient changes rapidly, wasting the optimization process that has been performed.

There are two methods of clipping: element-by-element for the parameter gradients g of a mini-batch and constraining the norm $||g||$ of the gradient, just before the parameter update [30]. The equation 3.1 is shown as clipping norm.

$$\begin{aligned} &if||g|| > v \\ &g \leftarrow \frac{gv}{||g||} \end{aligned} \tag{3.1}$$

Where v represents the threshold of the gradients norm.

3.1.2 Methods related to deep learning

Adam

Adam is short for Adaptive Moment Estimation. It is a popular optimization algorithm used in deep learning. It is an extension of the stochastic gradient descent(SGD) algorithm that uses adaptive learning rates for each parameter based on estimates of the first and second moments of the gradients [21].

Adam combines the advantages of two other popular optimization algorithms, Adagrads and RMSprop. Adagrad adapts the learning rate for each parameter based on the historical gradient information. RMSprop uses a moving average of the squared gradients to scale the learning rate. Adam also incorporates bias correction to account for the fact that the moving averages of the gradients are initially biased toward zero.

Adam is known empirically to be useful for training GAN [3].

Residual Nets

Residual Networks are also called ResNets. Although backpropagation enabled layer-by-layer learning, there was a problem as the layers got deeper, the gradient disappeared, and training did not progress. In response to this problem, a method was proposed to learn residuals from inputs by introducing a short-circuit connection 3.2 in which inputs are added in later layers [17]. This method prevents gradients from disappearing and enables models with more than 100 deep layers. In ResNet, each layer block

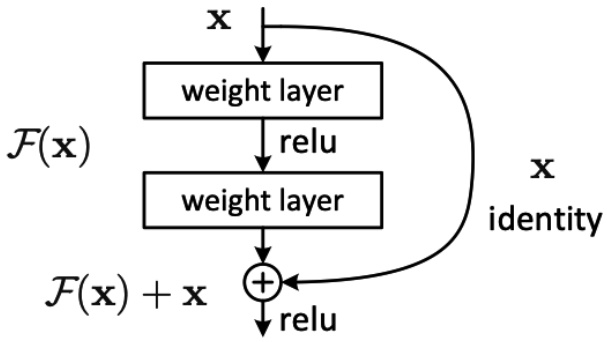


Figure 3.2: Residual Networks Block: image is retrieved from original paper [17]. Although backpropagation enabled layer-by-layer learning, there was a problem as the layers got deeper, the gradient disappeared, and training did not progress. In response to this problem, a method was proposed to learn residuals from inputs by introducing a short-circuit connection.

usually consists of a short-circuit connection that adds the output of that block to the input after several convolutional layers. The main advantage of ResNets is that they can be trained to very great depths, thus improving the accuracy of various tasks. ResNets are useful for image classification, object detection, and semantic segmentation tasks.

Convolutional Neural networks(CNN)

Convolution is one technique to improve the performance of full-connected neural nets, with the following motivations

1. Sparse connections
2. Parameter sharing
3. Equivalence representation

Convolution uses a window function called a kernel. This reduces the number of connections dramatically compared to a neural network fully connected, as only the connections in the applicable range of the kernel are applied. This saves memory and reduces the amount of computation. In addition, since it is compatible with parallel processing, it is possible to significantly shorten the learning time by parallel computation using Graphic Processing Unit (GPU), etc.

Since the kernels are applied in a moving manner, features that are trained commonly within the kernels, such as the edges of objects. Such parameter sharing allows the layer to be trained in equivalence representation.

Thus, while the model is computationally much less expensive, its performance in tasks such as image recognition is not degraded compared to the all-coupled model. It also has the advantage of avoiding over-fitting.

Although CNN is excellent at understanding local features, it is not good at understanding image stretching, rotation, and the positional relationship of each element within the whole. U-Net and Visual Attention will be explained later as methods to compensate for these.

U-Net

CNNs can grasp concepts at a high level of abstraction by convolution. On the other hand, the convolution lacks information on details. To compensate for this shortcoming, a U-Net was proposed that has a separate branch from the main trunk that performs convolution, bypassing detailed information and passing it to subsequent layers [12] [33].

The main branch that performs convolution grasps abstract concepts. The branches that are bypassed retain details. Combining these two allows both whole and partial information to be retained, aiming to

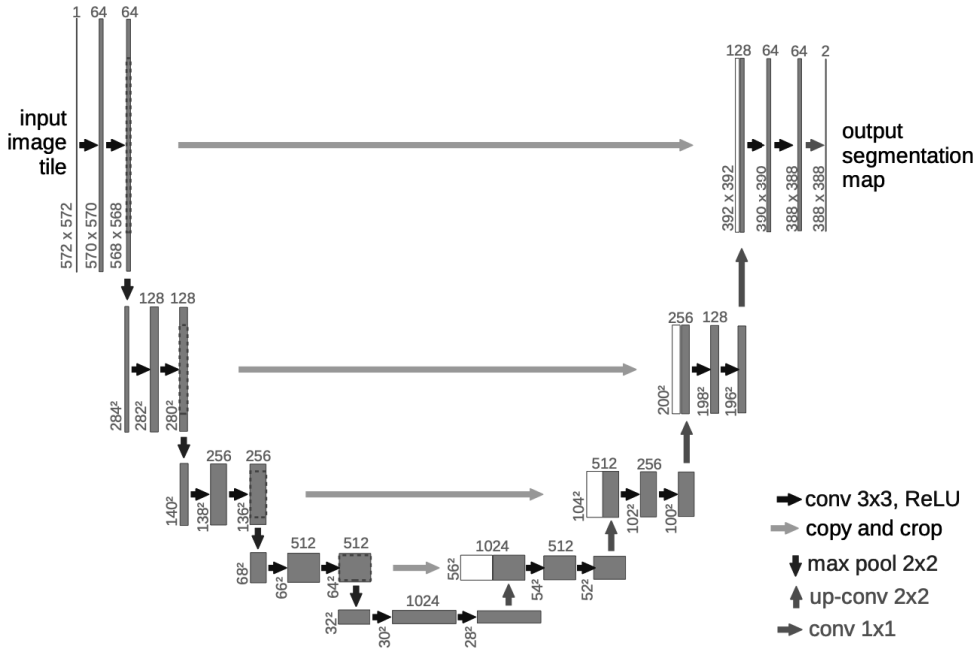


Figure 3.3: U-Nets: image is retrieved from original paper [33]. This figure shows a U-Net configuration for semantic segmentation of medical images. The image is compressed by max-pooling to extract information at higher levels of abstraction. At the same time, it aims to retain detailed information through bypassed output. This allows for better obtaining semantic segment and segment boundaries.

improve recognition accuracy. This is useful for tasks such as image recognition and semantic segmentation.

Figure 3.3 is the U-Net model proposed by O . Ronneberger et al. Gray arrows represent branches to be bypassed. Each is bypassed according to convolution depth and up-sampling.

Visual Self-Attention

Visual self-attention is a technique used in deep learning to process visual information. It uses a self-attention mechanism to selectively focus on different regions of an image, allowing the model to understand better the relationships between other parts of the image [27] [46].

The procedure for self-attention, figure 5.5 is to first find the inner product for a tensor (in this case, the image) with the transpose of that tensor. The result of this inner product is then applied to the softmax and multiplied by the tensor as a weight to be gazed at. Applying self-attention allows us to focus on different regions of the image based on their importance to a particular task. CNN excels at local feature understanding but has difficulty understanding the overall location of the features, improved with the application of self-attention. Visual self-attention has been used in various deep-learning models for tasks such as image captioning, object detection, and image segmentation.

Batch Normalization

In deep neural networks, during training, if the parameters or weights of one layer change, the distribution of inputs in the next layer changes. This reduces the learning speed. It also requires careful initialization of parameters and weights. Batch Normalization [20] was proposed to normalize the layer inputs to address these issues.

In recent deep learnings, mini-batches are generally used for training. In this method, normalization is performed for each mini-batch during learning. Precisely, the mean and standard deviation of the mini-batches are first calculated. Then, for each element of the mini-batch, the mean is subtracted and divided

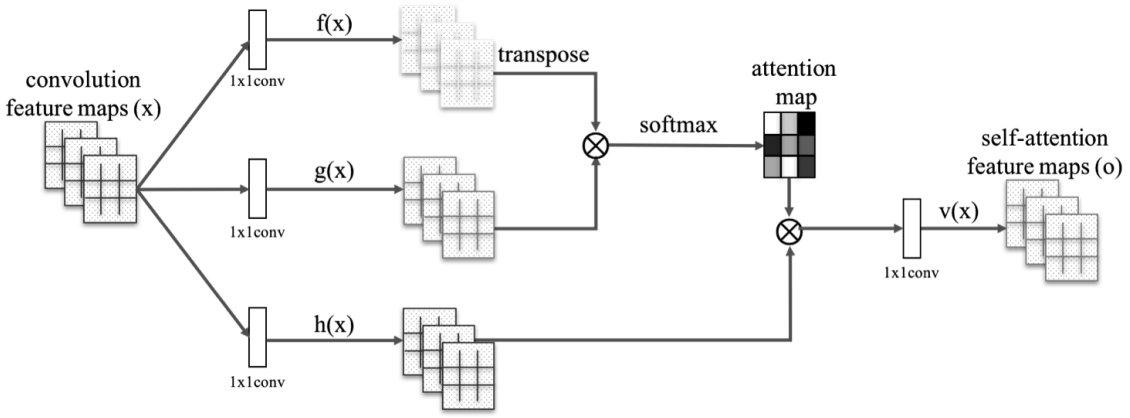


Figure 3.4: Visual Self-Attention: image is retrieved from original paper [46]. The procedure for self-attention is to first find the inner product for a tensor, yellow map, with the transpose of that tensor, green map. The result of this inner product that applied softmax as an activation function is called an attention map. Then, the attention map, as a weight to be gazed at, is multiplied by the input tensor, the purple map.

by the standard deviation. This procedure normalizes the mini-batch.

Normalizing the mini-batch speeds up the learning process. It also eliminates the need to care about the initial values of parameters and weights. No need to use Dropout [37]. Overfitting can be reduced. Relatively high learning rates can be used because learning is more stable. It also contributes to the learning stability of Generative Adversarial Networks (GAN). Because of these advantages, batch normalization is widely used in deep learning.

3.1.3 Generative Adversarial Networks (GAN)

Generative Adversarial Networks (GAN) are a type of deep learning architecture. The typical diagram is shown in Figure 3.5. It consists of two neural networks working against each other to generate realistic outputs. The first network, the generator, creates fake outputs from random values as input. The second network, the discriminator, distinguishes between generated fake and real data. They are called adversarial because they attempt to beat the other network through training. The generator and discriminator train alternately and can produce increasingly realistic outputs [15].

GAN has been applied to various tasks, including image and video generation, text generation, etc. It has also been used for data augmentation, style transfer, and anomaly detection. GAN's equation is shown in equation 3.2. Discriminator, D, and Generator, G, play a mini-max game with value function $V(G, D)$.

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log(1 - D(G(z)))] \tag{3.2}$$

As can be seen from the equation 3.4, the GAN discriminator uses binary cross entropy as the loss function since it is a binary True or False classification problem.

GAN training is prone to instability. Empirical methods are known to stabilize the training of GAN [3]. The following methods were also used in this study.

1. balancing the generator and discriminator
2. Using batch normalization
3. using hyperbolic tangent($tanh$) as the activation function

In GAN training, if one model is trained too quickly, the other model cannot keep up and causes mode collapse. To avoid this, it is necessary to balance the training speeds of the generators and discriminators.

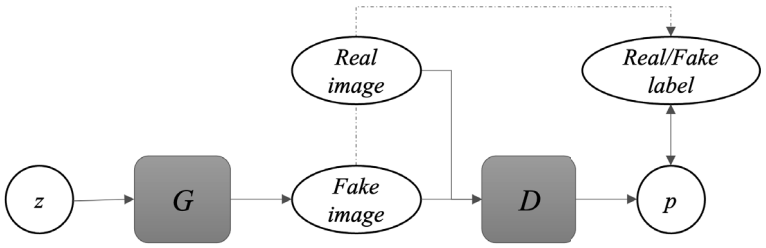


Figure 3.5: GAN has two neural network models, generator G and discriminator D. The discriminator tries to determine whether the inputs are real or fake and is trained with real/fake labels. The generator tries to fake the discriminator with fake outputs. They trained alternately in an adversarial way.

One way to do this is to make the geometry, number of parameters, and learning rate of the generator and discriminator models similar. This makes it relatively easy to adjust the training speed.

It is known that a moderate distribution of data across each layer between -1 and 1 is less likely to cause mode collapse, and thus, training is more likely to proceed. To achieve this, batch normalization is used to equalize the distribution of values. In addition, $tanh$ is used as the activation function, and for values exceeding ± 1, it is made to be within ± 1.

3.2 Directional Generative Networks(DGN)

3.2.1 Structure of DGN

Figure 3.6 shows DGN's model diagram. The method combines evaluation function F with deep learning models. The diagram of DGN is very similar to that of GAN. GAN and DGN differ in the built-in evaluation function F and constant values C as training direction. The model G's input z is random values, and the model G generates x.

The evaluation function F takes x and returns the result r. The estimator model E also takes x and returns the value p. The model E is trained from error between r and p. This makes the output of E gradually closer to the output of F. In other words, if E was trained enough, E is an approximate function of F.

For the training of model G, the model G and E are combined. The combined model is trained with error between target value C and predicted value p. When the combination model is trained, the weights of model E are fixed and not updated. By alternately training G and E, E increases the prediction accuracy, and G gradually becomes to be able to generate the preferred output.

The training procedure is shown in algorithm 1 The function mse at operation 8 and 9 in the algorithm 1 stands for mean squared error, which defined in equation (3.3).

$$mse(p, y) = \frac{1}{n}\sum_{i=1}^{n}(p_i - y_i)^2 \tag{3.3}$$

3.2.2 Comparison between GAN and DGN

DGN was conceived and inspired by the GAN learning method. A comparison of GAN and DGN is shown in the table 3.1 below.

For GAN, the input to the Generator is random values, and the input to the Discriminator is the output of the Generator as fake and real data from a pre-prepared dataset.

For the DGN input, random values are input to the generator. The output of the generator is input to the Estimator and Evaluation functions. A pre-prepared dataset is not necessarily required.

For output, the GAN's generator outputs fake data. The Discriminator outputs true/false label prediction to identify whether the received data is real contained in the dataset or fake data generated by the Generator.

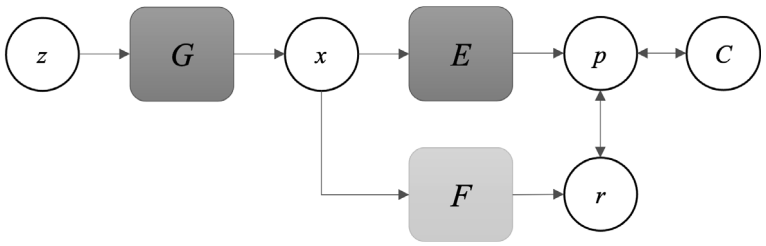


Figure 3.6: DGN Diagrams: The method combines evaluation function F with deep learning models. The diagram of DGN is very similar to that of GAN. GAN and DGN differ in the built-in evaluation function F and constant values C as training direction. The model G's input z is random values, and the model G generates x.
The evaluation function F takes x and returns the result r. The estimator model E also takes x and returns the value p. The model E is trained from error between r and p. This makes the output of E gradually closer to the output of F. In other words, if E was trained enough, E is an approximate function of F.
For the training of model G, the model G and E are combined. The combined model is trained with error between target value C and predicted value p. When the combination model is trained, the weights of model E are fixed and not updated.
By alternately training G and E, E increases the prediction accuracy, and G gradually becomes to be able to generate the preferred output.

Algorithm 1 Training procedure of DGN

1: Setting constant values C as the target direction
2: **for** number of iteration **do**
3: Gathering mini-batch with size m:
4: Randomly generate initial array $\left\{z \mid z^{(0)}, \cdots, z^{(m)}\right\}$
5: $\left\{x \mid G(z^{(0)}), \cdots, G(z^{(m)})\right\}$
6: $\left\{r \mid F(x^{(0)}), \cdots, F(x^{(m)})\right\}$
7: $\left\{p \mid E(x^{(0)}), \cdots, E(x^{(m)})\right\}$
8: Updating E with the mini-batch $\nabla_{\theta D}\left[\frac{1}{m}\sum_{k=0}^{m}[mse(r,\ p)]\right]$
9: Updating G concatenated fixed-weights E with the mini-batch $\nabla_{\theta G}\left[\frac{1}{m}\sum_{k=0}^{m}[mse(C,\ p)]\right]$
10: **end for**

Figure 3.7: Training Procedure of DGN: Setting constant values C as the target direction; operation 1. A mini-batch is prepared from the generated random number sequence z; operation 4. The mini-batch contains a random number sequence z, x generated with z as input; operation 5. Preparing r calculated by F with x as input; operation 6. Preparing p estimated by E with x as input; operation 7.
The parameters and weights of the predictive model E are updated from the errors of r and p; operation 8. The parameters and weights of the generative model G are updated from the error between C and p; operation 9. At this time, G and E are concatenated, and the weight of E is fixed. This means that only G is updated. The operations from 3 to 9 are iterated during training.

Like GAN, The DGN's generator takes a random value input and outputs some generated data. Instead of the GAN's discriminator, DGN uses an estimator and an evaluation function. The Evaluation function takes the output of the Generator as input and returns the value of the evaluation result. The Estimator takes the output of the Generator as input and outputs a value that predicts the output of the Evaluation function.

True/False labels are used for the generator and discriminator in GAN as the ground truth. In DGN, constant values as targets are used for the generator as the ground truth. For the estimator, outputs of the evaluation function are used as the ground truth. Here, ground truth refers to the correct answer data to be compared to check the predicted values in the machine learning model.

In GAN, the errors for backpropagation are calculated with True/False labels and the discriminator's outputs. In DGN, the errors for the estimator's training are calculated with the outputs of the evaluation function and the estimator's outputs. For the generator in DGN, the errors are calculated with the target constant values and the estimator's outputs.

Two models in GAN and DGN are trained alternately.

A comparison of training procedures using a block chart is shown in the figure 3.8.

3.2.3 Theoretical description

GAN's equation is shown again in equation 3.4. Discriminator, D, and Generator, G play mini-max game with value function $V(G, D)$[15]. The GAN discriminator is a binary classification problem that distinguishes between real and fake data. Therefore, Equation 3.4 is constructed based on binary-cross entropy.

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z \sim p_z(z)}[\log (1 - D(G(z)))] \tag{3.4}$$

For DGN, the probability distribution of the data in the GAN is replaced by the probability distribution in the space defined by the functions F. In DGN, minimize the values of $V_0(E)$ 3.5, and $V_1(G)$ 3.6. Because DGN is a regression model, the equation is based on the mean squared error (MSE) rather than the binary cross entropy in GAN.

$$V_0(D) = \mathbb{E}_{z \sim p_z(z)} \left[\frac{1}{n} \sum_{i=1}^{n} [E(G(z)) - F(G(z))]^2 \right] \tag{3.5}$$

$$V_1(G) = \mathbb{E}_{z \sim p_z(z)} \left[\frac{1}{n} \sum_{i=1}^{n} [E(G(z)) - c]^2 \right] \tag{3.6}$$

Even if the function F is not differentiable, the estimator $E(x)$ could approximate $F(x)$ like a differentiable function. In the not-differentiable case, the exploration space is often steep. In this case, some stabilizing methods like clipping-norm are required.

The generator G is trained with the error of the constant values c and the output r of F. After sufficient iterations, The generator G outputs $G(z)$ such that the output r of F is close enough to c. In other words, the generator can be trained with given constants as their targets.

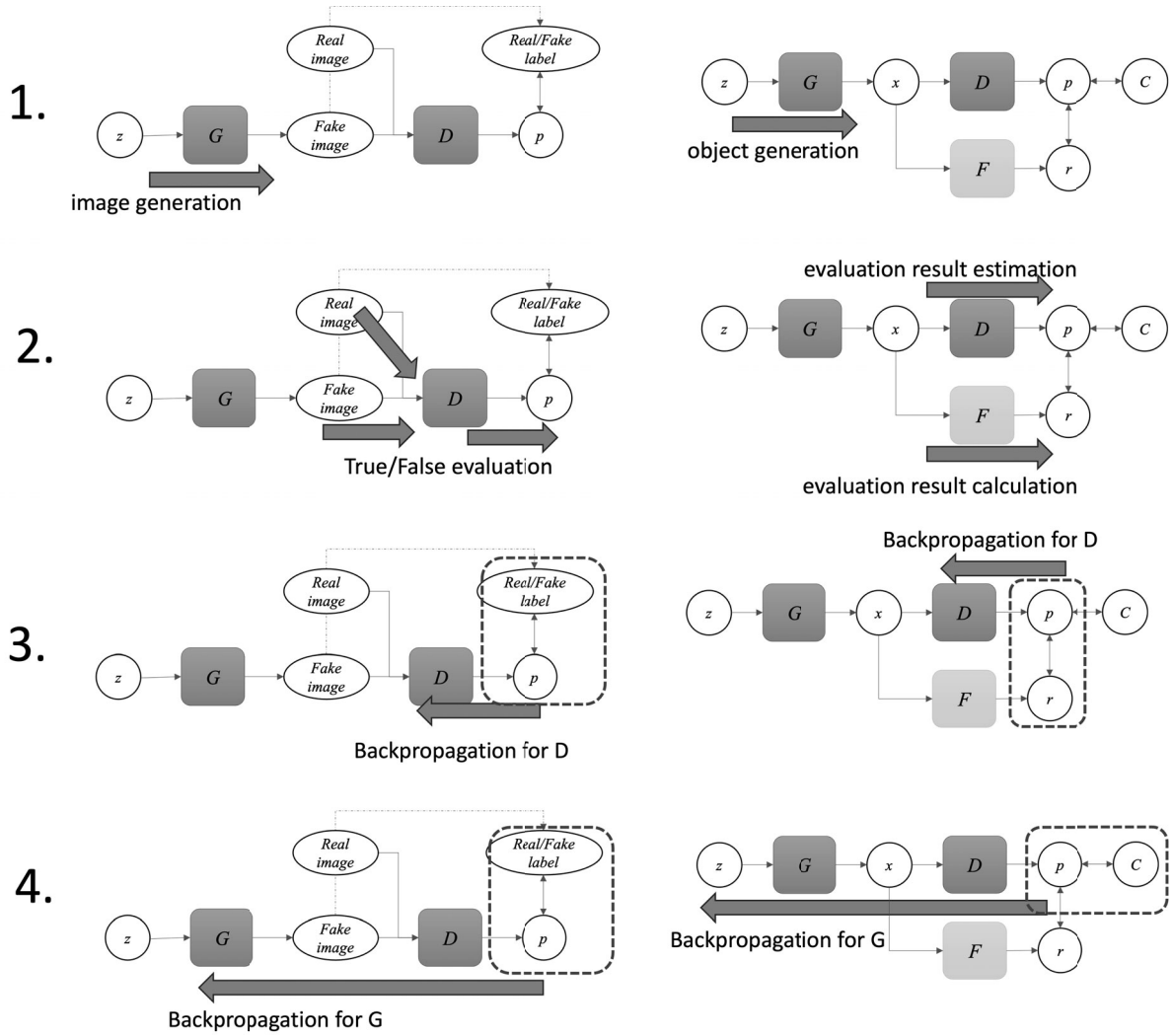


Figure 3.8: Comparison of training procedure:
The first step is generating objects or images from random input. This step is the same for GAN and DGN.
In the second step, the discriminator in GAN takes Real and Fake images. And predict whether the images are real or fake. In DGN, the estimator and evaluation function take the generated object x. The function outputs the evaluated result of the x. The estimator model estimates the evaluated result of the x.
In the third step in GAN, the error is calculated using the Real or Fake labels and predicted labels. Then, the weights of the discriminator are updated by the error with the backpropagation. In the third step in DGN, the error is calculated with the evaluated results and predicted values, Then, the weights of the estimator are updated by the error with the backpropagation. Therefore, if the estimator is trained enough, the estimator becomes an approximate function of the evaluation function.
In the fourth step in GAN, the weights of the generator are updated by the error with the back-propagation. At that time, the discriminator's weights are not updated. In the fourth step in DGN, the error is calculated with the predicted values by the estimator and constant values. The constants are the target of the object x. the constant is the ideal value for the object x. Then, the weights of the generator are updated by the error with the backpropagation. At that time, estimator D is also not updated like GAN.
These four steps are iterated in the training phase until the models are trained enough. If the models in DGN are trained enough, the evaluated result of generated x is close enough to the target constant values. DGN is a cooperative model rather than adversarial.

	GAN	DGN
Inputs	**For Generator** :Random values **For Discriminator** Fake(Generator's output) and real data	**For Generator** :Random values **For Estimator and Evaluation function** : Generator's outputs
Outputs	**From Generator**: Fake data, **From Discriminator** : True/False estimations	**From Generator** : Generated data, **From evaluation function** : evaluated values for generated objects **From Estimator**: Estimated values for the evaluation function outputs
Ground truth	**For Generator and Discriminator** : True/False labels	**For Generator**: Constant values as the target, **For Estimator**: Outputs of the evaluation function
Error	**For Generator and Discriminator** : error are calculated with True/False labels and discriminator's outputs	**For Estimator**: error are calculated with outputs of the evaluation function and Estimator's outputs **For Generator**: error are calculated with Constant target values and Estimator's outputs
How to train	Generator and Discriminator are alternately trained	Generator and Estimator are alternately trained

Table 3.1: Comparison between GAN and DGN: Input for GAN is random values for the generator, the output of the Generator as fake, and real data from a pre-prepared dataset for the discriminator. Input for DGN is random values for the generator. The output of the Generator is input to the estimator and Evaluation functions. A pre-prepared dataset is not necessarily required.

Output of GAN, the generator outputs fake data. The discriminator outputs true/false label prediction to identify whether the received data is real contained in the dataset or fake data generated by the generator. Output of DGN: The generator outputs some generated data. The evaluation function returns the value of the evaluation result based on the generator's outputs. The estimator takes the output of the Generator as input and outputs a value that predicts the output of the evaluation function.

As ground truth, True/False labels are used for the generator and discriminator in GAN. In DGN, constant values as targets are used for the generator. For the estimator, outputs of the evaluation function are used.

The error calculation for backpropagation, In GAN, the errors are calculated with True/False labels and the discriminator's outputs. In DGN, the errors for the estimator's training are calculated with the outputs of the evaluation function and the estimator's outputs. For the generator in DGN, the errors are calculated with the target constant values and the estimator's outputs.

Two models in GAN and DGN are trained alternately.

3.3 Simple Examples of DGN

Here are two Equations 3.7 and 3.8 as examples of the function F in DGN. Equation 3.7, used in the first example, is a simple differentiable function that returns the sum over all three arguments and the product over all three arguments. The expression 3.8 for the second example is a non-differentiable function that returns the remainders of the first and second arguments and the remainders of the second and third argument. Constants in Equation 3.9 are given as the target for each example.

$$F_0(x_0, x_1, x_2) = \left[\sum_{k=0}^{2} x_k,\ \prod_{k=0}^{2} x_k \right] \tag{3.7}$$

$$F_1(x_0, x_1, x_2) = [x_0 \bmod x_1,\ x_1 \bmod x_2] \tag{3.8}$$

$$C = [0.5,\ 0.035] \tag{3.9}$$

Training procedure of the simple example

The input z to the generative model is random numbers. The training procedure below follows the previous pseudo code 1.

1. For the output $[x_1, x_2, x_3]$ of the generative model, obtain the return value $[y_0, y_1]$ of the function F_0 or F_1 that takes it as an argument, and the output $[y'_0, y'_1]$ of the estimation model D_0.
2. From the $[y_0, y_1]$ and $[y'_0, y'_1]$, obtaining the loss function values. The loss function is Mean-Squared Error (MSE) Equation 3.10.
3. Update the weights of E from the loss function value.
4. $[y'_0, y'_1]$ obtained from the model that combined G and E.
5. The loss function values are obtained from $[y'_0, y'_1]$ and the target constants $C = [0.5, 0.035]$.
6. Update the weights of G from the loss function values. Here, the weight of E connected to G is fixed and is not updated.

By repeating the above procedure, the estimation model E output gradually closes to the output of the function F_0 or F_1. Also, the output $F(G(z))$ gradually closes to the target constant due to the generator G's weights update.

$$mse(p, y) = \frac{1}{n} \sum_{i=1}^{n} (p_i - y_i)^2 \tag{3.10}$$

Result

Figure 3.9 shows the loss values transition of the E model during training. The x-axis is the iteration number of training. The y-axis is the loss value. The total iteration number is 2^{20}. Figure 3.10 shows loss values transition of combination model, G and D_1 while training. All of them are trained well.

Calculated values $[y_0, y_1]$ transition with generated values $[x_0, x_1, x_2]$ are shown in Figure 3.11 and 3.12. As in the Loss values figure, the line colors, orange and grey, are examples with the function F_0, and blue is an example with F_1. Target values are $[0.5, 0.035]$. Therefore, $[y_0, y_1]$ are converged well.

Figure 3.13 on left shows generated values $[x_0, x_1, x_2]$ with trained model G in example with function F_0. The red color is x_0, the green is x_1, and the blue is x_3. Figure 3.13 on right hand shows caluculated values $[y_0, y_1]$ with $[x_0, x_1, x_2]$. The calculated values $[y_0, y_1]$ are well converged. Nevertheless, the generated $[x_0, x_1, x_2]$ maintain diversity. It can be seen that a variety of values are generated that realize Target's values.

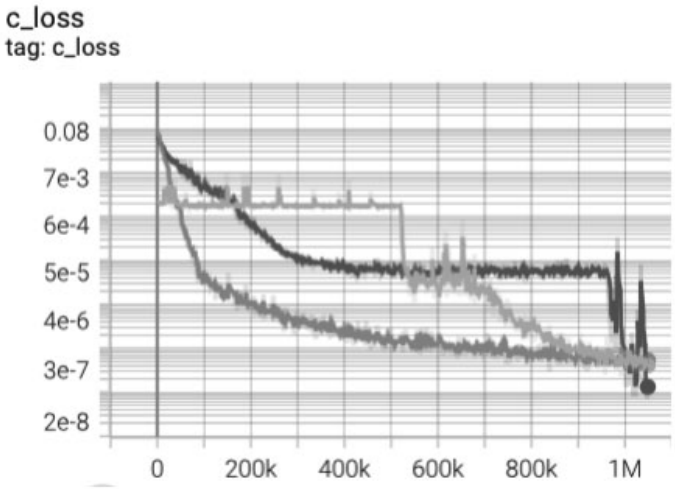


Figure 3.9: Loss value transition of estimator model; Simple Examples of DGN: The x-axis is the iteration number of training. The y-axis is the loss value. The total iteration number is 2^{20}.

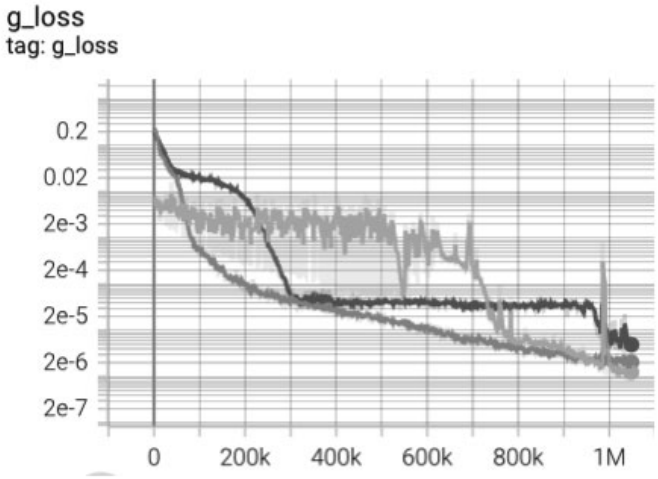


Figure 3.10: Loss value transition of the combined model; Simple Examples of DGN: which combined with G and E. The x-axis is the iteration number of training. The y-axis is the loss value. The total iteration number is 2^{20}. Line colors orange and grey are examples of the function F_0, and blue is an example of F_1.

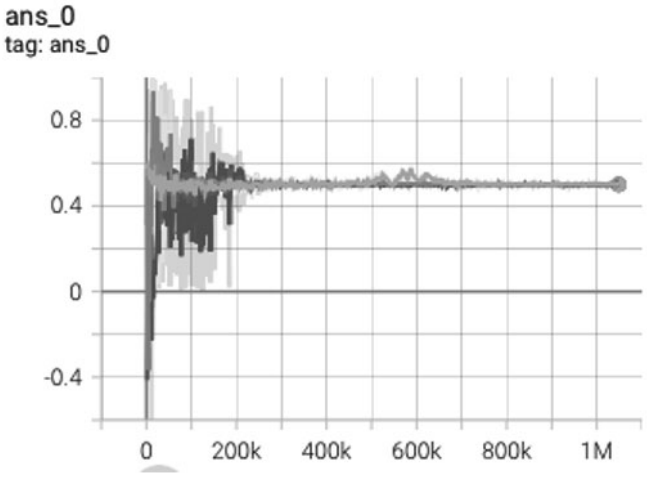


Figure 3.11: Calculated value y_0 transition with generated values $[x_0, x_1, x_2]$ The target value is 0.5. Line colors, orange and grey are examples with the function F_0, and blue is an example with F_1. Each experiment shows good convergence to the target value.

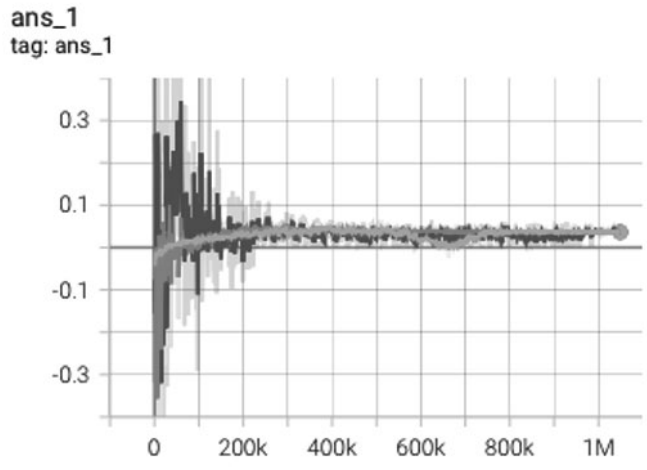


Figure 3.12: Calculated value y_1 transition with generated values $[x_0, x_1, x_2]$ Target values are 0.035. Line colors, orange and grey are examples with the function F_0, and blue is an example with F_1. Each experiment shows good convergence to the target value.

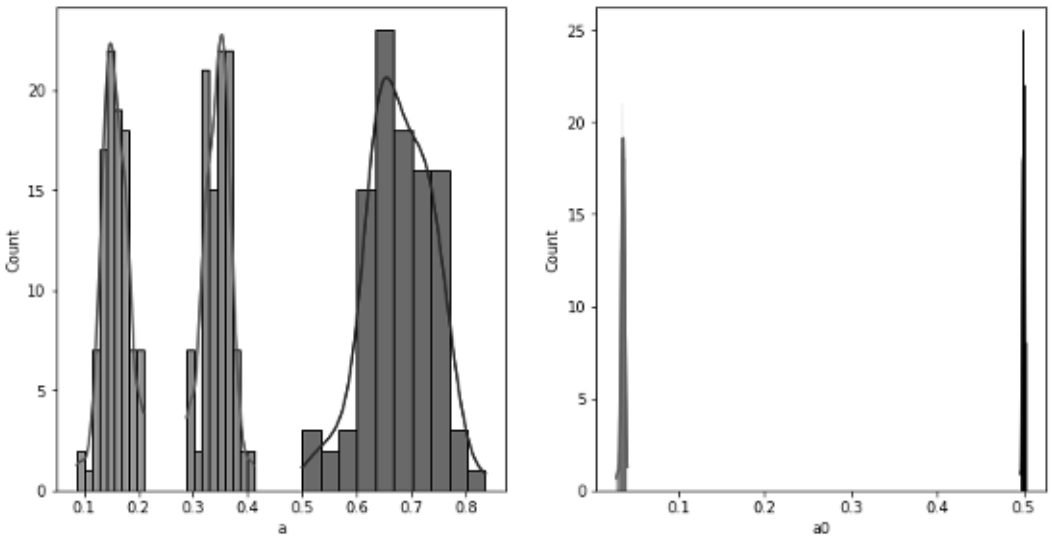


Figure 3.13: Distribution of generated values for the function F_0, using addition and multiply: On left shows generated values $[x_0, x_1, x_2]$ with trained model G in example with function F_0. The red color is x_0, the green is x_1, and the blue is x_3. Figure 3.13 on right hand shows caluculated values $[y_0, y_1]$ with $[x_0, x_1, x_2]$. The calculated values $[y_0, y_1]$ are well converged. Nevertheless, the generated $[x_0, x_1, x_2]$ maintain diversity. It can be seen that a variety of values are generated that realize Target's values.

Figure 3.14 on the right is an example of function F_1, with values $[x_0, x_1, x_2]$ generated using the learned model G. Red is x_0, green is x_1, and blue is x_3. The figure on the right 3.14 shows the calculated value $[y_0, y_1]$ with $[x_0, x_1, x_2]$. The calculated value $[y_0, y_1]$ converges well too. However, the generated $[x_0, x_1, x_2]$ are still diverse, but not as diverse as the example of the function F_0. This is expected because the function F_1 is non-differentiable, which means that it was difficult to train around the point where the values are discontinuous. However, since the calculated values converge well, we can say that the models E approximate the function F_1 well in the range where the value is generated with the model G.
These examples are elementary and can be described as toy problems. Real-world problems are much more complex. Functions representing many problems are often non-differentiable. However, if a good approximation can be made to the extent that it is useful from an engineering standpoint, it can be an effective tool. This study examines the applicability of this tool to real-world problems in chapter 4 and chapter 5.

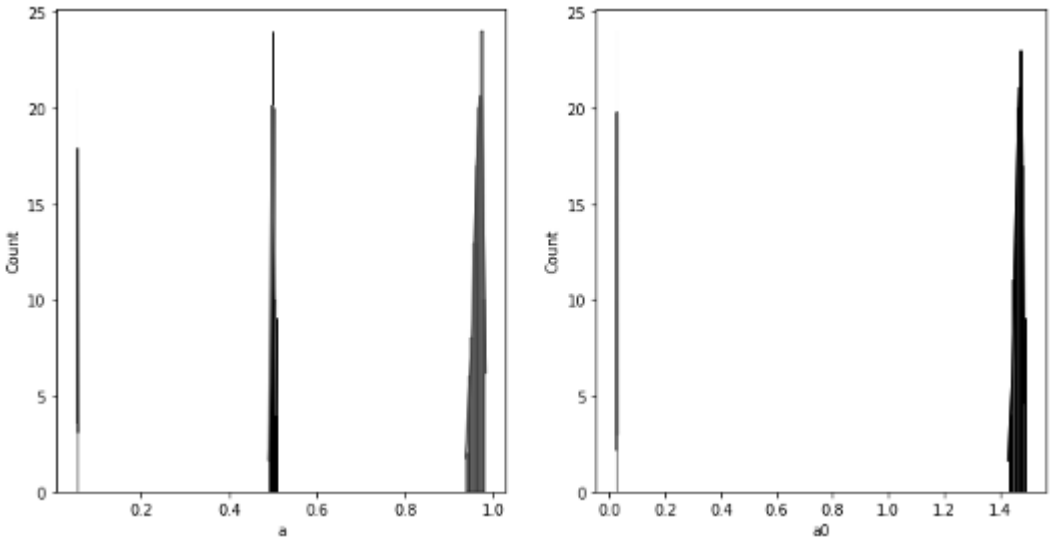


Figure 3.14: Distribution of generated values for modulo function F_1: On the right is an example of function F_1, with values $[x_0, x_1, x_2]$ generated using the learned model G. Red is x_0, green is x_1, and blue is x_3. The figure on the right 3.14 shows the calculated value $[y_0, y_1]$ with $[x_0, x_1, x_2]$. The calculated value $[y_0, y_1]$ converges well too. However, the generated $[x_0, x_1, x_2]$ are still diverse, but not as diverse as the example of the function F_0. This is expected because the function F_1 is non-differentiable, which means that it was difficult to train around the point where the values are discontinuous. However, since the calculated values converge well, we can say that the models E approximate the function F_1 well in the range where the value is generated with the model G.

Chapter 4

DGN Application for Structural Design

In this chapter, DGN is applied to building structural design. Building data is much more complex than the simple examples described in chapter 3. The application applies DGN to the shear panels' arrangement, which is structural members. If the shear panels can be appropriately placed based on the building data without structural data, the DGN is considered to have the functionality to support structural design.

The structural designer cannot immediately determine whether the design meets the requirements simply by visually reviewing the structural design drawings. To make a judgment, it is necessary to check the output results of the structural calculations. Based on the output results, the structural design is changed, and the structural calculation is repeated in an iterative process. Appropriate placement of shear panels is mainly evaluated based on the eccentricity and Demand capacity ratio(DCR) values, which are structural safety check items specified in the Building Standard Act. In other words, if the values of eccentricity and DCR are appropriate for the arrangement of shear panels generated, DGN can be considered a helpful tool for engineering design support.

It is common for there to be more than one appropriate structural member arrangement for the building data. However, the available data is expected to be only one of the adopted design alternatives. Therefore, only a narrow search space may be learned if the shear panel arrangement is given as the ground truth. In this study, instead of using the shear panel arrangement data corresponding to the architectural design data, the values of eccentricity and DCR, which are structural safety check items, are fed back for learning. The purpose is to obtain a model that can generate various solutions.

In this chapter, the applicability of DGN is examined. Section 4.2 describes the structure, data representation, and training procedure of the DGN for structural design. Section 4.3 describes the measurement metrics, data representation, and training/ testing results as an experiment. Section 4.4 summarises this Chapter.

The description in this chapter is limited to what is described in the patent application or the public domain. Matters relating to trade secrets are not discussed here and are omitted.

4.1 Structural Design

4.1.1 Structural Design

Structural design in civil engineering involves the following steps in general [36].

1. **Project Planning:** In this phase, the project team identifies the project's requirements, objectives, and constraints, including site conditions, client needs, budget, and relevant regulations.

2. **Preliminary Design:** The structural engineer develops preliminary design concepts based on the project requirements and constraints.

3. **Analysis** Once the preliminary design is complete, the engineer performs a detailed analysis of the structure to ensure that the structure complies with relevant codes and other performance requirements.

4. **Design Development** The analysis results refine the preliminary design and produce detailed design drawings, including plans, elevations, sections, and details.

5. **Review and Approval** The design is reviewed and modified as necessary by a project team that includes the client, architect, and other interested parties. Plans are then submitted to the appropriate authorities, such as the building department and regulatory agencies, for approval. Construction Once the design is approved, the construction phase begins. Engineers may provide construction support services such as site visits and inspections to ensure that the structure is constructed as designed.

6. **Post-construction evaluation** After construction, the engineer performs a post-construction review to ensure the structure meets design specifications and functions as intended. This may include load testing and other evaluations.

Because construction projects are generally large in scale, design work may be subdivided into a hierarchy. For example, the design work can be divided into basic design, in which overall requirements or concepts are studied to be satisfied, and detailed design, in which each part is designed to satisfy the needs of the basic plan. Evaluation is generally conducted in each of these phases. Overall, the structural design process in civil engineering involves a cooperative effort among the project team, engineers, and other stakeholders to ensure the structure is safe, efficient, and meets the client's needs.

4.1.2 Domain Knowledge

The domain knowledge is essential to apply machine learning methods or other systems to support design processes. Here, domain knowledge refers to the knowledge and know-how specific to an industry sector. For example, the designer of a building structure must have the knowledge and skills to understand the following technical backgrounds and set appropriate constraints according to the client's requirements. After several years of training, the designer is finally ready for design work.

1. **Knowledge of materials and material mechanics:** Typical materials used in building structures include concrete, steel, and wood. Each material has inherent properties such as stiffness, allowable stress, weight, and anisotropic/isotropic properties. Ease/difficulty of processing and ease/difficulty of obtaining materials differ depending on the material, and the designer must select the appropriate material according to the client's requirements.

2. **Knowledge of building construction methods** Since buildings are assembled at the construction site, transporting, assembling, and joining the components are essential. For steel construction, high-tension bolted or welded joints are commonly used. The designer must properly design the joint locations of the structure to divide the members into appropriately sized sections that can be loaded onto trucks delivered to the site.

3. **Knowledge of loads acting on buildings:** Loads working on a building during its service life can be divided into long-term and short-term loads. Long-term loads are the dead weight of the building itself and the weight of the people and fixtures that use it. Short-term loads include seismic loads, wind loads, and snow loads. The designer must design the structure with an appropriate safety buffer for each load risk.

4. **Knowledge of Geotechnical and Soil Mechanics:** Buildings are constructed on the ground. Improper ground investigation or foundation design can lead to building settlement or tilt, resulting in significant losses. Therefore, it is necessary to understand the behavior of the soil under building loads, e.g., consolidation settlement, liquefaction during earthquakes, and foundation pile behavior. The soil or sand that makes up the ground is a highly variable material, and the designer must properly design the foundation to account for this variability.

5. **Analysis Methods:** It is necessary to understand how the designed building structure will be deformed and what internal forces will be generated in response to external forces such as the

abovementioned seismic forces. The finite element method has been commonly used for analysis in recent years. In Japan, elastoplastic analysis is sometimes used to take advantage of the structure's ability to absorb energy through plastic deformation to avoid over-designing for the possibility of massive earthquakes. Designers should understand the technical background of these analysis methods and use them appropriately.

6. **Regulations and Standards for Building Structure:** Most buildings are private assets, but they are also social assets. If a building were to collapse due to a moderate earthquake that could occur during its service life, the social loss would be enormous. From the viewpoint of public welfare, minimum standards are set by law. Materials used in construction must also be by Japanese Industrial Standards and other standards. Designers must design in compliance with these standards.

7. **Knowledge of basic building design:** Communication between the structural designer and architect is essential. During the structural design process, plans are frequently adjusted. When design changes occur due to the client's intention, etc., the feasibility is examined from the viewpoint of structural design. Even if feasible, the designer must also consider cost-effectiveness since achieving this at a reasonable cost may not be possible. Communicating to accommodate such changes requires general architectural knowledge, not just structural design.

4.2 DGN for Structural Design

4.2.1 Model Structure

The model structure of the DGN is shown in Figure 4.1. In this section, seismic forces are considered external forces, and a design that meets the requirements of the Building Standard Act is considered. Here, we consider a structure where the shear panels bear the most horizontal forces. Let F_0 be a function of this model for the tool to evaluate the Proposed design. This function F_0 takes the building structure as input and returns its eccentricity and Demand Capacity Ratio (DCR). Eccentricity and DCR are explained in Appendix, Chapter 7.

As an intuitive explanation of this model, the generative model G on the left in the figure is a model for generating a design (generated data) with the desired performance, and estimation model D_0 on the right side of the figure is a model for estimating the result of the function F_0. If the estimation model D_0 is sufficiently trained, it becomes an approximate function of the function F_0. When the generative model G is trained, it is connected to the estimated model D_1 with fixed weights, and G is trained with the error between the output of $D_1(G(d, z))$ and the fixed value C, that is the desired value.

By training these models appropriately, the quality of the generated design is expected to improve. In other words, the evaluation result of the building structure generated by G will be closer to the desired result, C. For example, if you want 0.95 for the demand-capacity ratio of the building structure, set 0.95 at the corresponding position in C.

G: Generator, is neural network-based model that generates x, shear-panel locations with grey-scale images. The input of G has two parts: an initial state that randomly generated z and a building data d. The building data consists of each floor image made with floor-slab images, outer and inner wall images, and windows and doors images. F_0 is a function that takes x as input and returns demand-capacity ratio (DCR) and eccentricity measurements. These measurements are described in Chapter 7 section 7.1.

G, Generator and D_0/D_1, Estimator of $F_0(x)$ have been based on U-net [33] structure, and combined with self-attention [27] block. D_0 and D_1 receive x, the generated shear-panel images. D_0 is trained with the output of r, $F_0(x)$ as ground truth. Thus, p, the output of D_0 gradually approaches the output of $F_0(x)$. In other words, if D_0 had trained enough, it acts as an approximate function of $F_0(x)$. The model D_1 is exactly the same as D_0. D_1's only difference with D_0 is not trainable. When learning G, the weights of D_1 are fixed when updating the weights of G. c is constant target values, desired values of measurements.

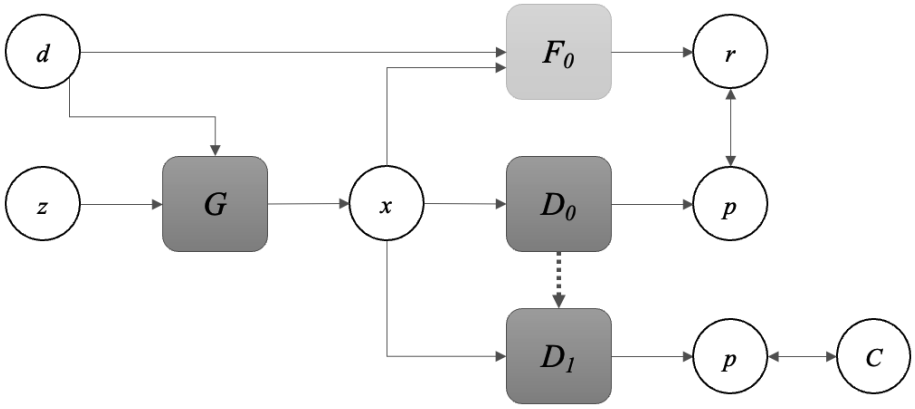


Figure 4.1: DGN Training Scheme for Structural Design:

4.2.2 Data Representaion

The data used for training and testing is that of a built house. In general, the CAD (Computer Aided Design) data consists of the name of each element and the coordinate values at which that element is located. Each of these elements is treated as an image in a two-dimensional array, with one stored in the array location corresponding to the coordinates at which it is located 4.2.

A certain standard length, for example, one meter or one foot, is equivalent to one pixel in an image. An image can be described as a multidimensional array. For example, if a structural member is present at a specific location, one is stored at the corresponding position in the array, and if not present, 0 is stored at that location. The image size of the training data and the generated shear panel images are 64 pixels by 64 pixels. Each image deals with a horizontally projected member of each floor. Mirror flipping and data rotation (0°, 90°, 180°, 270°) were used as data enhancement to increase the number of data.

The generated image of the sheared panel is shown in Figure 4.3. The generated shear panels are separated in the x- and y-axis directions. These images show the recommended locations of the shear panels to resist horizontal forces in each direction, in the x-axis direction for the second floor. Locations where it is desirable for shear panels to be placed generate larger values, while undesirable locations generate smaller values. In the figure, darker colors represent larger values, and lighter colors represent smaller values. Locations where shear panels cannot be placed are white as 0.

This data format makes it easier to apply CNN. Even if the size of the building changes, the same model can be used for training/generation as long as the building size fits within 64 pixels x 64 pixels.

4.2.3 Training

The following pseudo-code 2 shows the training procedure for DGN. As the hyperparameter on the raining, the number of training iterations is 35000.

Where, F_0 is a function that takes d, building data and x, the generated shear panels as input, and returns the calculated result values such as demand-capacity ratio and eccentricity.

The function $logcosh$ at operation9 and 11 in the algorithm 2 is defined in equation (4.1).

$$logcosh(p, y) = \log\left(\frac{e^{p-y} + e^{-p+y}}{2}\right) \tag{4.1}$$

These models cooperate; they gradually approach the desired solution. As the prediction accuracy of the computational results around the search area improves, more favorable sheare panels arrangement will be generated.

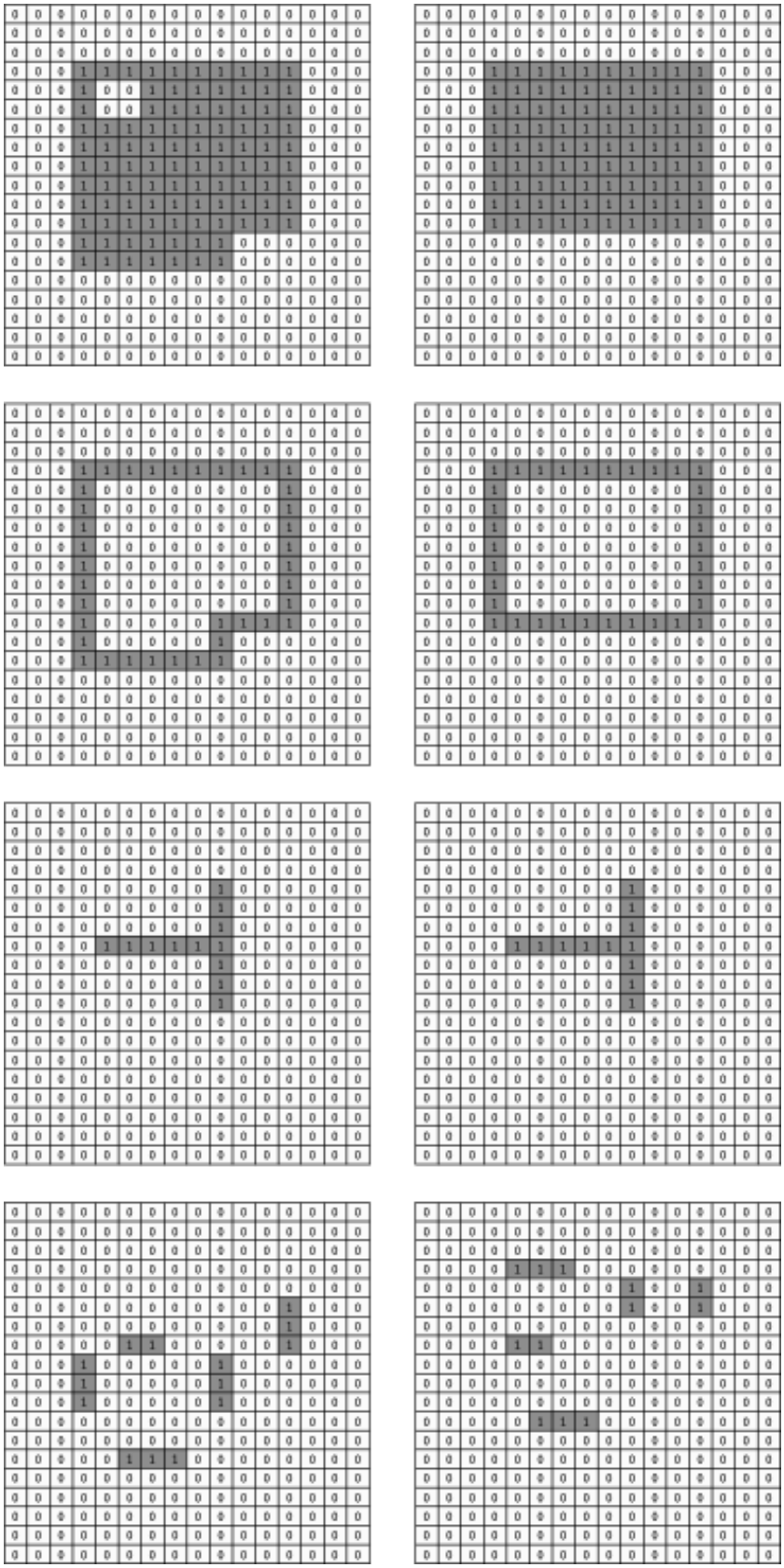


Figure 4.2: Building Data Representation Simplified Examples: On the 1st row, the left is the 2nd-floor slab; the right is the roof slab. On the 2nd row, the left hand is the outer wall of the 1st floor, and the right hand is the outer wall of the 2nd floor. On the 3rd row, the lefthand is the inner wall of the 1st floor, and the right hand is the inner wall of the 2nd floor. On the 4th row, the left hand is doors and windows on the 1st floor, and the right hand is doors and windows on the 2nd floor. Each array element contains "1" if the building part exists and "0" if it does not.

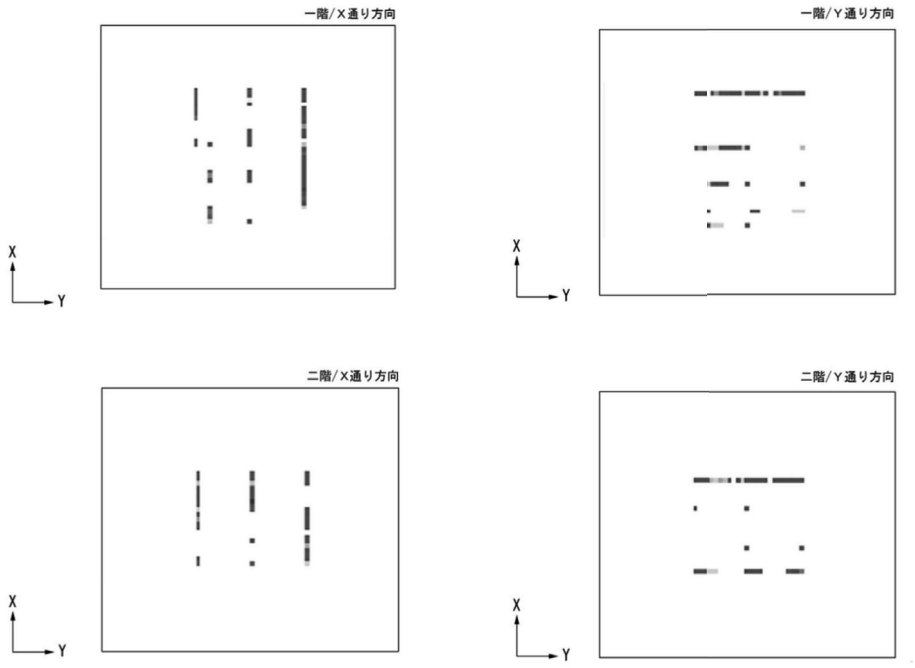


Figure 4.3: Data Representation of Generated Shear-Panels: The generated shear panels are separated in the x- and y-axis directions. Upper left is the y-axis direction of the first floor, top right is the x-axis direction of the first floor, top left, in the y-axis direction on the second floor; top right, in the x-axis direction on the second floor; The top right shows the recommended locations of the shear panels to resist horizontal forces in each direction, in the x-axis direction for the second floor.
Locations where it is desirable for shear panels to be placed generate larger values, while undesirable locations generate smaller values. In the figure, darker colors represent larger values, and lighter colors represent smaller values. Locations where shear panels cannot be placed are white as 0.

Algorithm 2 Training Procedure D_0, G

1: Setting constant values C as the target direction
2: **for** number of iteration **do**
3: Minibatch Preparation: Sample minibatch of m samples
4: $d = \left[d^{(1)}, \ldots, d^{(m)}\right]$ from building data.
5: $z = \left[z^{(1)}, \ldots, z^{(m)}\right]$ from gaussian noise.
6: $x = \left[G\left(d^{(1)}, z^{(1)}\right), \ldots, G\left(d^{(m)}, z^{(m)}\right)\right]$
7: $r = \left[F_0\left(d^{(1)}, x^{(1)}\right), \ldots, F_0\left(d^{(m)}, x^{(m)}\right)\right]$
8: $p = \left[D_1\left(d^{(1)}, x^{(1)}\right), \ldots, D_1\left(d^{(m)}, x^{(m)}\right)\right]$
9: Updating D_0 with the mini-batch by stochastic gradient:
$\nabla_{\theta D_0}\left[\frac{1}{m}\sum_{k=0}^{m}\left[logcosh(r,\ p)\right]\right]$
10: Copying weights from D_0 to D_1
11: Updating the weights of generator G with the mini-batch by stochastic gradient:
$\nabla_{\theta G}\left[\frac{1}{m}\sum_{k=0}^{m}\left[logcosh(C,\ p)\right]\right]$
12: **end for**

Figure 4.4: Training Procedure; DGN for Structural Design: Setting constant values C as the target direction 1. Prepare A mini-batch of m samples; operation 3. For mini-batch preparation, building data samples are gathered from training data; operation 4. Generate random number sequence z; operation 5. Generate shear panels arrangement x with generator G from d and z as inputs; operation 6. Calculate measurements results r with function F from x as input; operation 7. Estimate measurements results p with the estimator model D_1 from x as input; operation 8.
The weights of the predict-model D_0 are updated from the errors of r and p; operation 9. Copying weights from trainable model D_0 to non-trainable model D_1; operation 10. The weights of the generate-model G are updated from the error between C and p; operation 11. At this time, G and D_1 are concatenated, and the weight of D_1 is fixed. This means that only G is updated. The operations from 3 to 11 are iterated during training.

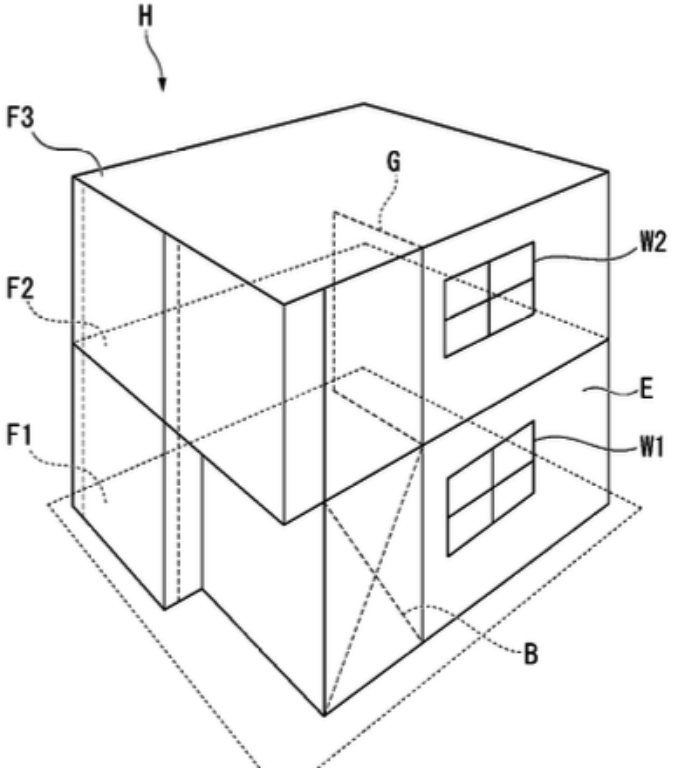


Figure 4.5: Building Data Elements: F1, F2, and F3 in the image show 1st floor slab, 2nd floor slab, and roof slab, respectively. E is outer-wall. G is inner wall. B shows doors. W1 and W2 are windows.

4.3 Experiment

4.3.1 Measurments

Eccentricity and Demand capacity ratio are used as measurements.

1. **Eccentricity** The eccentricity ratio indicates the building's tendency to twist based on the distance between the center of gravity and the stiffness center of the building. If the eccentricity ratio is large, the building will twist when horizontal forces are applied, and structural design considerations are required to resist such twisting. For example, if the eccentricity exceeds 0.15, the horizontal force for structural design should be increased during an extreme earthquake. The smaller the eccentricity, the more desirable it is, but it is challenging to make it completely zero.

2. **Demand Capacity Ratio** The demand capacity ratio (DCR) is the ratio of the stress generated to the allowable capacity value of a structural member. If the DCR is greater than 1, the member is considered unsafe. If the DCR is below 1, it is considered safe, but if it is significantly below 1, it is of excessive quality. Therefore, the closer to 1, but not exceeding 1, the more desirable it is.

The calculation methods for eccentricity and DCR are shown in the Appendix, Chapter 7. For example, as the target, the constant values could be set at 0.0 as the eccentricity and 1.0 as DCR.

4.3.2 Method

As inputs to the DGN model, the geometry of the building's exterior walls, interior walls, windows, doors, and floors, and the weight of each part of the building in figure 4.5, are given. For this input, the presence or absence of each element is stored as 0 or 1 in an array of 64 by 64 pixels.

The building data were divided into training data and test data in the experiment. Neither the training data nor the test data contained the correct shear panel placement. The shear panels were generated

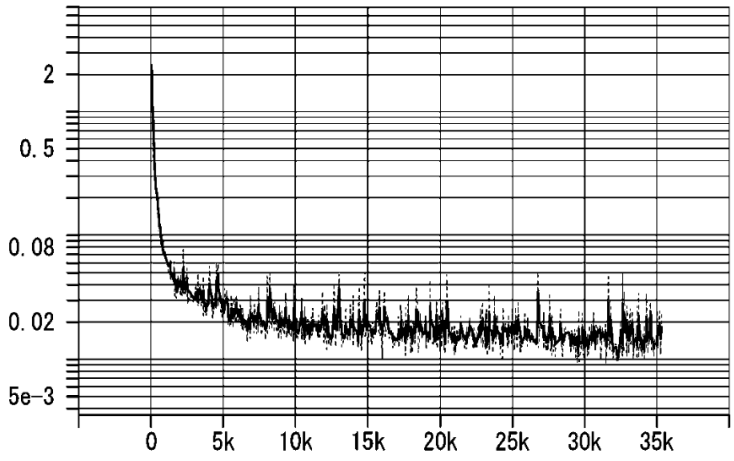


Figure 4.6: Loss values transition of calculation result estimator, D: During the training, estimators' loss value is stably decreased.

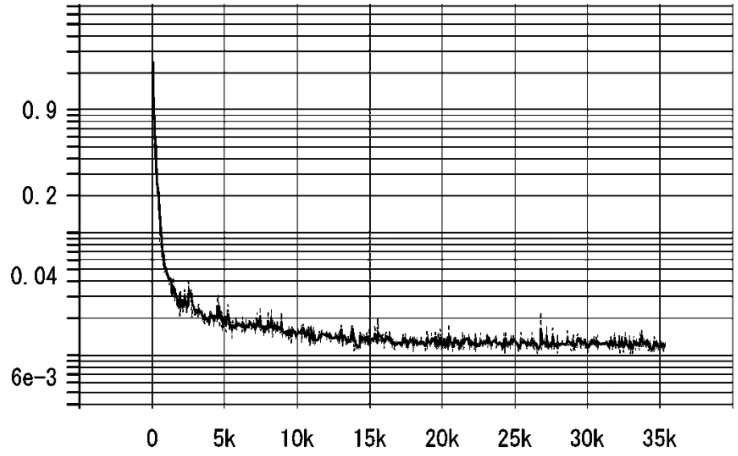


Figure 4.7: Loss values during training of shear panel generator, G: The model is combined with genarator G and fixed-weights estimator D_1. Like loss values, the loss value of the combined model is stably decreased.

from the test data as a test in an iterative procedure within the pseudocode that illustrated the training procedure. Therefore, the number of training iterations coincided with the number of test iterations. The sheared panel arrangement generated from that test data was used as input to the function F, and the eccentricity and DCR were obtained as output. The obtained eccentricity and DCR trends were used to evaluate the training success or failure of the training.

4.3.3 Result

Figure 4.6 and Figure 4.7 below are the loss function values of models D_0 and $D_1(G)$ at training. Figure 4.8 is the loss function value of $D_1(G)$ with test data. The horizontal axis indicates the number of training iterations. The vertical axis shows the loss function value in that training iteration.

From this, we can see that models D_0 and $D_1(G)$ are being trained successfully, with the loss function values decreasing as the training progresses. Testing is being conducted in parallel with training. The loss function value of $D_1(G)$ from the test data is generally larger than that from the training data. There is a temporary phase in which the loss function value increases, but in general, the loss function value decreases as learning progresses, indicating that generalization ability has been acquired.

During the training and testing, the eccentricity and DCR transition obtained from the shear panel arrangement generated by model G from the test data as input to the function F_0 are shown in Figure 4.9

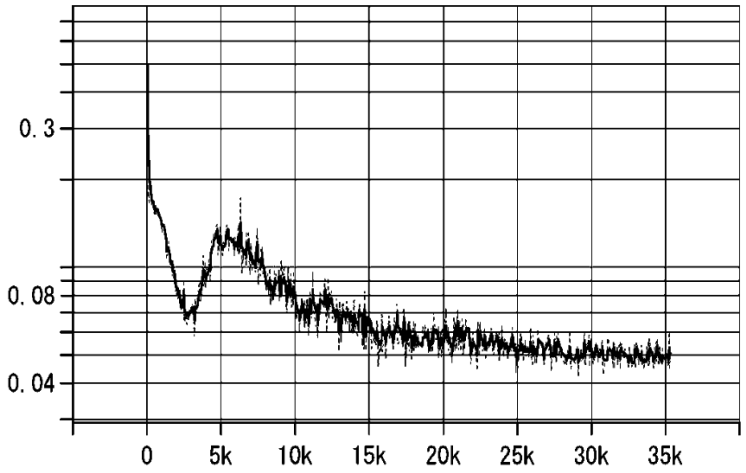


Figure 4.8: Loss values of G with test data: Up to about 2.5k iterations, G and D are learned separately and are stable. Between 2.5k and 5k iterations, the loss function value is expected to deteriorate due to the competition between G and D. After 5k iterations, the loss function value begins to decrease again. Naturally, the loss function values from the test data are larger than those from the training data. However, it can be said that a certain level of generalization ability has been acquired. Test data refers to the building plan data not included in the training data. Training and test data do not contain the correct shear panel arrangement data.

and Figure 4.10. For eccentricity, the values are generally around 0.2, although there are some outliers. The DCR also has some outliers but is generally between 0.6 and 1.0. Therefore, most shear panel arrangements are appropriately designed as building structures.

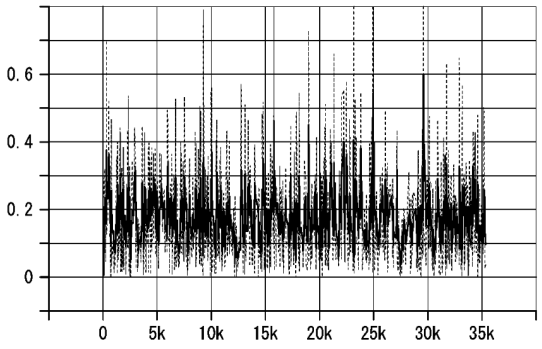
4.4 Summary of the Chapter

It seems feasible to apply DGN to building structural design. When DGN was applied to the design of shear panel placement in architecture, it was found to be adequately trained. In addition, its learned generative model acquired a certain generalization capability. The eccentricities and DCRs calculated from the shear panel placement generated from the test data were generally adequate, although there was some variation.

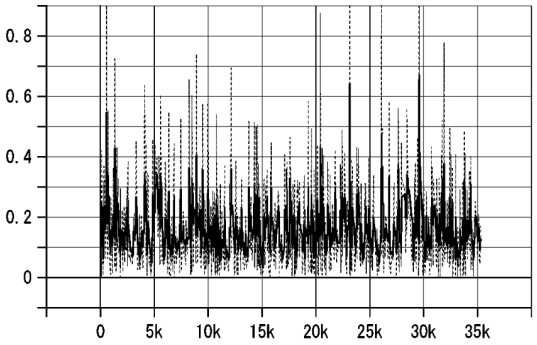
The DGN was used to design the placement of shear panels in a building structure, and the model was found to be adequately trained. After repeated mini-batch training of approximately 35,000 iterations, it was confirmed that the loss function values of the models that inferred the structural calculation results converged below a certain value. The loss function values of the model that concatenates the generated model with the fixed weights of the model that guesses the structural calculation results also converged well. It was confirmed that even a regression problem, rather than a GAN-like model that only guesses true-false values, can be trained without divergence.

The learned generative model of DGN acquired a certain generalization ability. During the learning process, we used test data not included in the training data to check the transition of loss function values. Although there was a temporary increase in the loss function value, the loss function value continued to fall steadily after that. Naturally, the test data's loss function values exceeded the training data's loss. Still, they did not diverge or overfit, and a constant generalization capability was obtained.

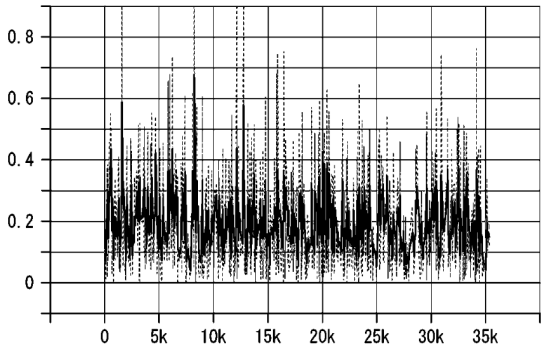
The eccentricities and DCRs calculated from the shear panel configurations generated from the test data were generally adequate, albeit with some outliers. When the DGNs' generator model is used as a design aid tool, multiple shear panel placement patterns are generated from buildings' architectural data. Of those multiple generated shear panel placement patterns, the patterns containing outliers are discarded, and the preferred one is selected from the remaining designs. Used in this way, it can be a helpful tool for engineering purposes.



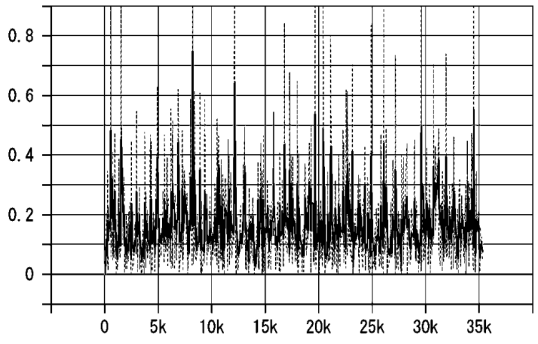
(a) Eccentricity, 1st Floor, X-axis



(b) Eccentricity, 1st Floor, Y-axis



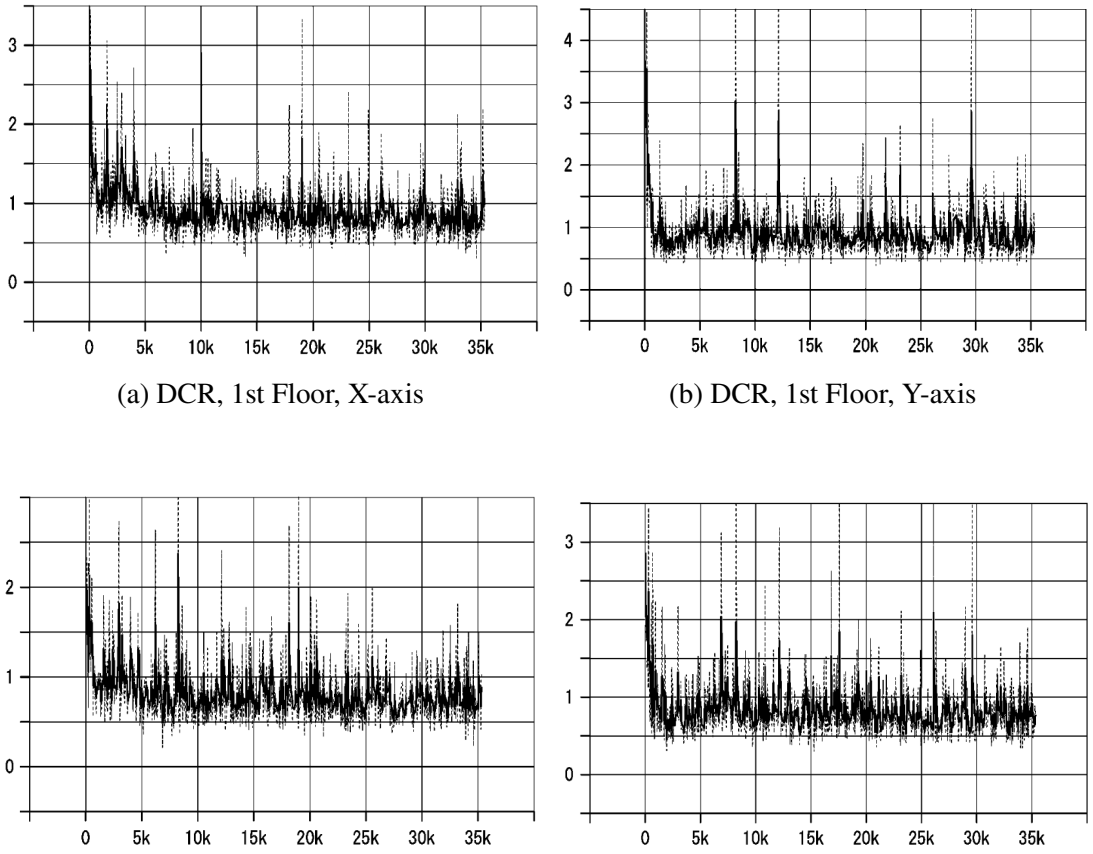
(c) Eccentricity, 2nd Floor, X-axis



(d) Eccentricity, 2nd Floor, Y-axis

Figure 4.9: Eccentricity transition with test data during training: The values are generally around 0.2, although there are some outliers. Multiple shear panel arrangements are generated for actual use as a design aid tool. Among the generated shear panel arrangements, those that contain outliers in the output of the evaluation function F are eliminated, and the preferred one is selected from the remaining arrangements. In doing so, it becomes a helpful tool for engineering purposes.

The DGN was found to be adequately trained to place shear panels in building structures. In addition, its learned generative model acquired a certain generalization capability. The eccentricities and DCRs calculated from the shear panel arrangements generated from the test data were generally adequate, although there was some variation. Based on the above, applying DGN to building structural design seems feasible.



(a) DCR, 1st Floor, X-axis

(b) DCR, 1st Floor, Y-axis

(c) DCR, 2nd Floor, X-axis

(d) DCR, 2nd Floor, Y-axis

Figure 4.10: Demand Capacity Ratio(DCR) transition of shear panels with test data during training: The DCR also has some outliers but is generally between 0.6 and 1.0. Multiple shear panel arrangements are generated for actual use as a design aid tool. Among the generated shear panel arrangements, those that contain outliers in the output of the evaluation function F are eliminated, and the preferred one is selected from the remaining arrangements. In doing so, it becomes a helpful tool for engineering purposes.

Chapter 5

DGN Application for Molecular Search

In this chapter, the applicability of DGN for molecular search is examined. This study shows that a generative model trained in DGN can generate molecules with the desired characteristics by feeding back the measurements calculated with the evaluation function. DGN is compared to ORGAN and MolGAN in this chapter. ORGAN and MolGAN were already discussed in chapter 2.

DGN is also compared to evolutionary algorithms(EA) here. When the problem complexity increases, there is a case where the EA performance decreases. EA performs random searches based on good individuals. DGNs based on deep learning perform searches according to the gradient of the search space. Therefore, DGNs are expected to work on complex problems where evolutionary algorithms do not work well if the model has sufficient expressive capability.

Section 5.1 shows the background of molecular search tasks. Section 5.2 describes the structure of DGN for molecular search, data representation, and the theoretical description of the learning procedure. Section 5.3 describes experiments, including metrics and comparison of generated molecules with EAs. Section 5.4 summarizes the chapter.

5.1 Molecular Search

Drug discovery, which contributes to people's health, is one of the most important fields of research and development. Huge budgets and time tend to be consumed in the process of drug discovery. Discovering new molecules, e.g., candidates for new drugs, is still a challenging task due to the vastness of the search space [31]. In addition to physical discovery using high-throughput screening(HTS) and other methods, recent advances in computational power have led to the experimentation of various virtual discovery methods. Computational methods to obtain new molecules with desirable characteristics can be roughly classified into the following categories. Naturally, attempts are also being made to improve performance by combining these methods.

1. **Virtual Screening:** Virtual Screening(VS) [32] methods include Substructure search, Similarity, Docking, and QSAR [35]. Substructure search method searches for molecules that contain the desired substructure in a given database of molecules. Similarity method searches a database of molecules for molecules similar to the molecule with the desired property. Docking methods search the database for molecules that fit the target enzyme or receptor. QSAR stands for Quantitative Structure-Activity Relationship. The QSAR method searches for molecules with predicted activity from the database based on the activity data of the compound.

2. **Randomized algorithms:** Evolutionary Algorithms(EA) is well known as a typical randomized algorithm [18]. The advantage of EA is that it can be applied even if the differentiability of the evaluation function is not certain. Genetic Algorithm(GA) is one of the EAs proposed by Holand. J. [18]. A unique aspect of GA is its emphasis on crossovers. In searching for new molecules with desirable features, Lee, Y. et al. proposed a method of stacking two GA models [26]. This is intended to focus on a desired region of the search space. The method uses the similarity to

a specific molecule prepared from the data set and the desired features as a score for individual selection.

3. **Generative models:** Generative Models based on Deep Learning [14], Variational Autoencoder(VAE) [22] is known as a likelihood-based generative model, Samanta et al. applied VAE to the graph structure of molecules. [34] Generative Adversarial Networks(GAN) [15] as a likelihood-free generative model. While likelihood-free models have disadvantages compared to likelihood-based models, such as low explanatory power and difficulty in learning, they are known to have advantages, such as the ability to learn even objects with complex data structures.

As the generative models, ORGAN [16] and MolGAN [9] can generate desirable molecules.

This model uses multiple discriminators to learn the correctness of molecules and generates a group of molecules with desirable properties as a GAN. This makes it possible to generate molecules with favorable scores for Drug-likeness [6] logP [43] [7] and Synthesizability [10]. However, depending on the dataset used for learning, the variation of molecules generated may be limited. If the number of data used to train a GAN is insufficient, the output may be identical to those in the data set used for training. In MolGAN, molecules with specific characteristics are selected and used as training data, Therefore, it is likely that the same molecules as those used for training are generated, resulting in a smaller Uniqueness score.

Most of the methods rely on the datasets of molecules. Since the molecules obtained can vary depending on the data set used for the search, it would be difficult to eliminate the influence of bias in the data set. Due to the difficulty of generating molecules with reasonable structures, few studies have attempted to do so without relying on datasets. In this study, we attempted to construct a training-data-free generation model by combining modules for chemical informatics functions with the model. The proposed method performed better than EA for more complex tasks.

Molecular searches are challenging, physically realistic, and virtual searches due to the sheer volume of combinatorial patterns. Many datasets have been proposed that consist of molecular structures and measurements of the molecules. However, a dataset is a subset of a vast search space and cannot cover everything. In addition, preparing a new dataset is not easy due to the human and financial resources required.

As already mentioned, evolutionary algorithms are computationally expensive for complex systems. If the number of solution combinations becomes too large, they will not converge to a suitable solution due to the explosion of combinations. Multiple GA models were stacked in the EA/GA applications already seen. Molecular search requires domain knowledge to compress the search space because of the vast number of combination patterns.

In addition, most other machine learning methods rely on existing datasets. Therefore, it is difficult to remove the bias of that dataset. Also, acquiring molecules outside the space defined by the dataset is generally impossible.

5.2 DGN for Molecular Search

5.2.1 Model structure

Directional Generative Networks (DGN) have almost the same structure as GANs. However, DGN does not require training data for molecular search. By not using a dataset, we aimed to eliminate the bias caused by the dataset. When a generative model is properly trained with training data, the model has generalization ability within the search space defined by the training data. In other words, if the search space is vast and the available training data is limited, the generalization ability of the model will depend on the data, and it will not have the ability to generate models for other search spaces. Therefore, we aimed to create a generative model that does not use any training data.

The model structure of DGN is shown in Fig. 5.1. G: Generator, is neural network-based model that generates SMILES [42] with one-hot encoding. The input of G has two parts: a few initial atoms that are randomly generated and a mask based on the atom number of the molecule. F_0 is a function that takes x, the generated SMILES as input, and returns the SMILES with the grammatically incorrect parentheses

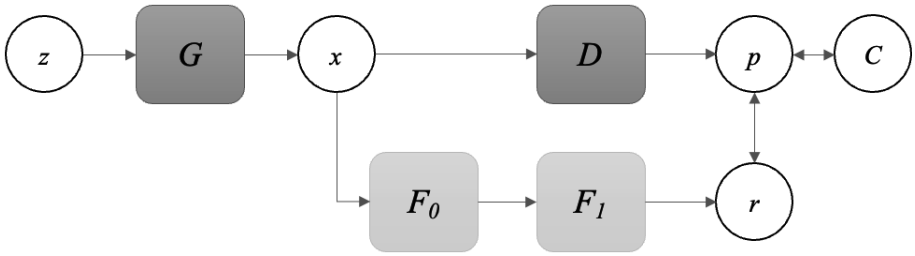


Figure 5.1: DGN Structure. G, D: Estimator of Calculation model D receives x, the generated SMILES as input. D learns with the output of $F_1(F_0(x))$ as ground truth. Thus, the output of D gradually closes the output of $F_1(F_0(x))$. When learning G, the weights of D are fixed when updating the weights of G. c is constant for the target measured values as ground truth.

and ring numbers corrected as output. F_1 is a function that include the RDKit [25] module, an open source toolkit for chemical informatics. F_1 takes the generated SMILES as input and returns the QES, SAS, and logP measurements. These measurements are described in subsection 5.3.1.
Two methods based on domain knowledge were used in DGN: masking the positions of sequences that exceed the required number of atoms by a factor of 0 and using the function F_0 to convert them into a form suitable for the SMILES grammar.

G, Generator and D, Estimator of $F_1(F_0(x))$ have been based on U-net [33] structure, and combined with self-attention [27] block. D receive x, the generated SMILES. D is trained with the output of r, $F_1(F_0(x))$ as ground truth. Thus, p, the output of D gradually approaches the output of $F_1(F_0(x))$. In other words, if D had trained enough, it acts as an approximate function of $F_1(F_0(x))$. When learning G, the weights of D are fixed when updating the weights of G. c is constant target values and desired molecular features. For example, if you want 0.6 for the QED value of the molecule to be generated, set 0.6 at the corresponding position in C.

Generator

The generator model G is shown in figure 5.2. There are two inputs to the Generator. One input z_0 is an array containing a few randomly selected atoms. The second input, z_1, is an array that represents the number of atoms in the molecule and acts as a mask. The number of randomly selected atoms in the first input and the range of randomly chosen numbers in the second input are specified in advance as hyperparameters.

The first and second inputs are concatenated and normalized in the Batch Normalization layer. In the next convolution layer, they are adjusted to the specified number of filters and concatenated into an Unet Block and a Self Attention block.

Sigmoid is used as the activation function to represent the type of atoms that consist of the molecule as One-hot-encoding. Finally, the second inputz_1 is multiplied to set the range where the molecule does not exist to 0.

The details of the generator G is shown Figure 7.1 7.2 7.3 7.4 7.5 in Appendix.

Estimator

The estimator D_0 and D_1 are shown in figure 5.3. The input to Estimator receives the generated molecules expressed in one-hot encoding.

The first layer of convolution is adjusted to the specified number of filters. Next, it is connected to the self-attention block. The next layer is connected to the U-Net Block. The order of self-attention and U-Net is reversed in the generative model.

The output of the U-Net branches into three branches. The output of each branch is scalar, with a full-connection layer. The activation function is not used for the outputs to make it a regression problem.

The details of the estimator D_0 and D_1 is shown Figure 7.6 7.7 7.8 7.9 7.10 in Appendix.

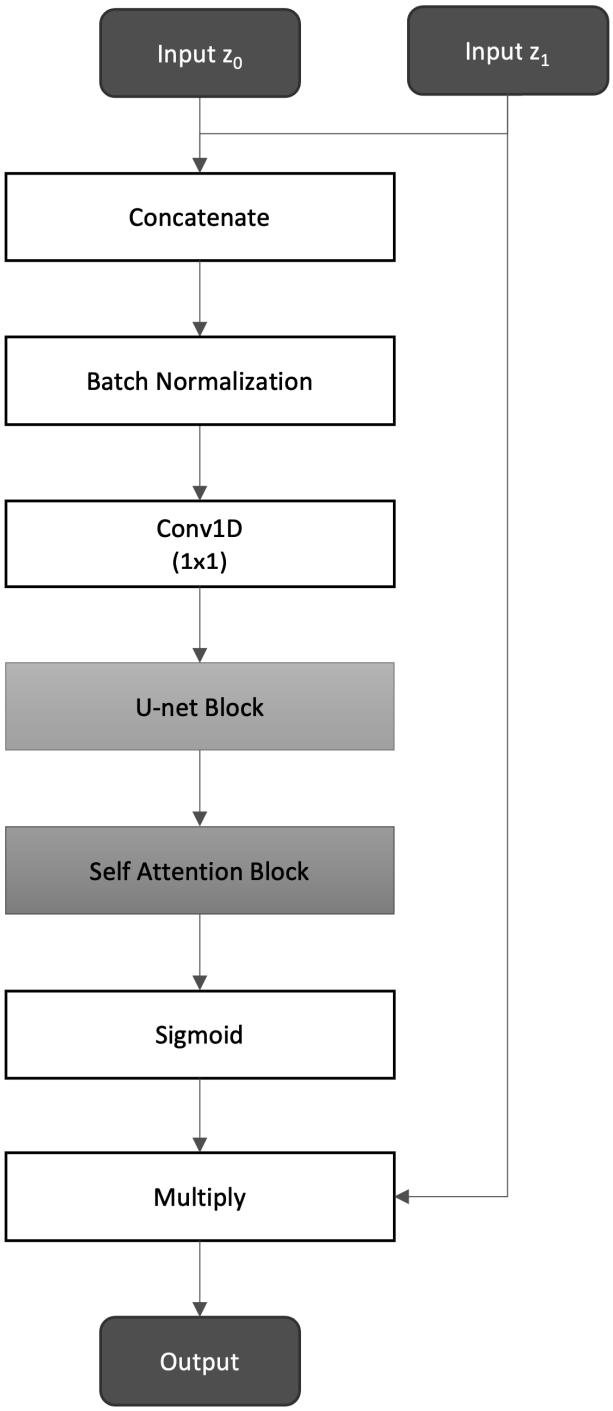


Figure 5.2: Generator: One input z_0 is an array containing a few randomly selected atoms. The second input, z_1, is an array that represents the number of atoms in the molecule and acts as a mask. The number of randomly selected atoms in the first input and the range of randomly chosen numbers in the second input are specified in advance as hyperparameters.
The first and second inputs are concatenated and normalized in the Batch Normalization layer. In the next convolution layer, they are adjusted to the specified number of filters and concatenated into u-net and self-attention blocks.
Sigmoid is used as the activation function to represent the type of atoms that consist of the molecule as One-hot-encoding. Finally, the second inputz_1 is multiplied to set the range where the molecule does not exist to 0.

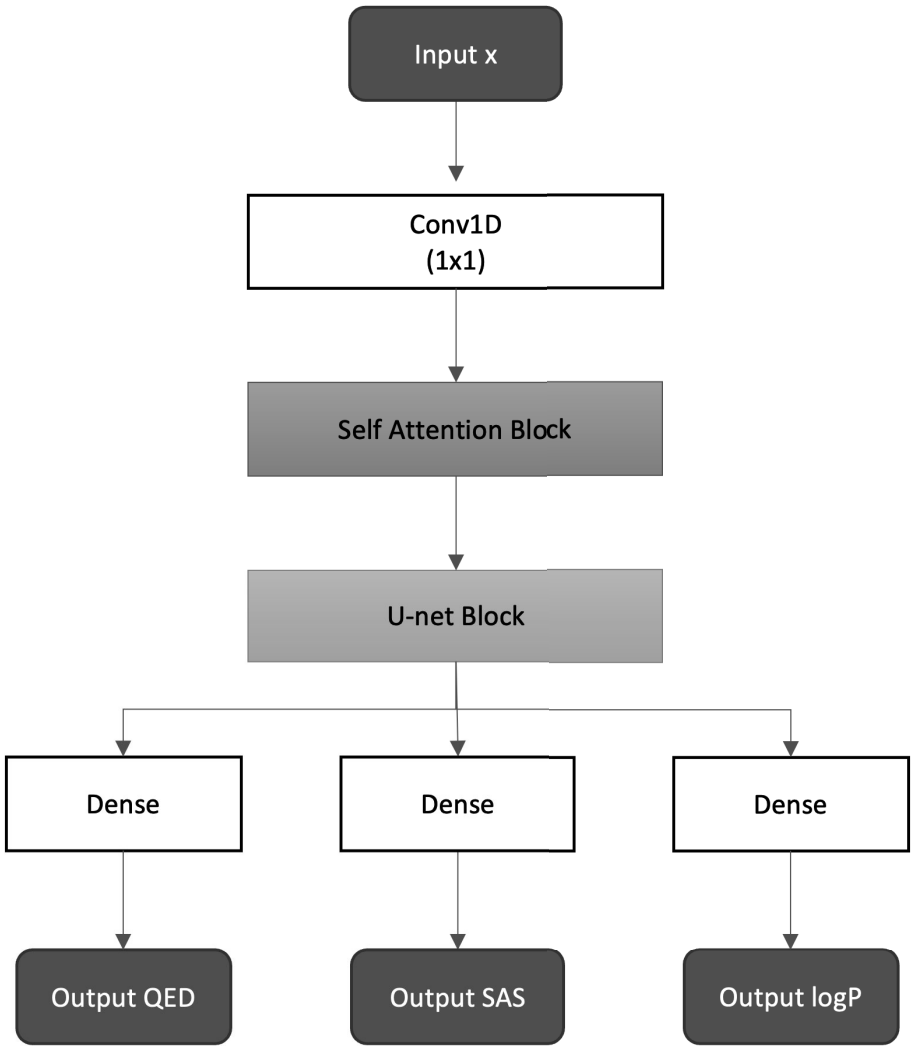


Figure 5.3: Estimator: The input to Estimator receives the generated molecules expressed in one-hot encoding.
The first layer of convolution is adjusted to the specified number of filters. Next, it is connected to the self-attention block. The next layer is connected to the U-Net Block. The order of self-attention and U-Net is reversed in the generative model.
The output of the U-Net branches into three branches. The output of each branch is scalar, with a full-connection layer. The activation function is not used for the outputs to make it a regression problem.

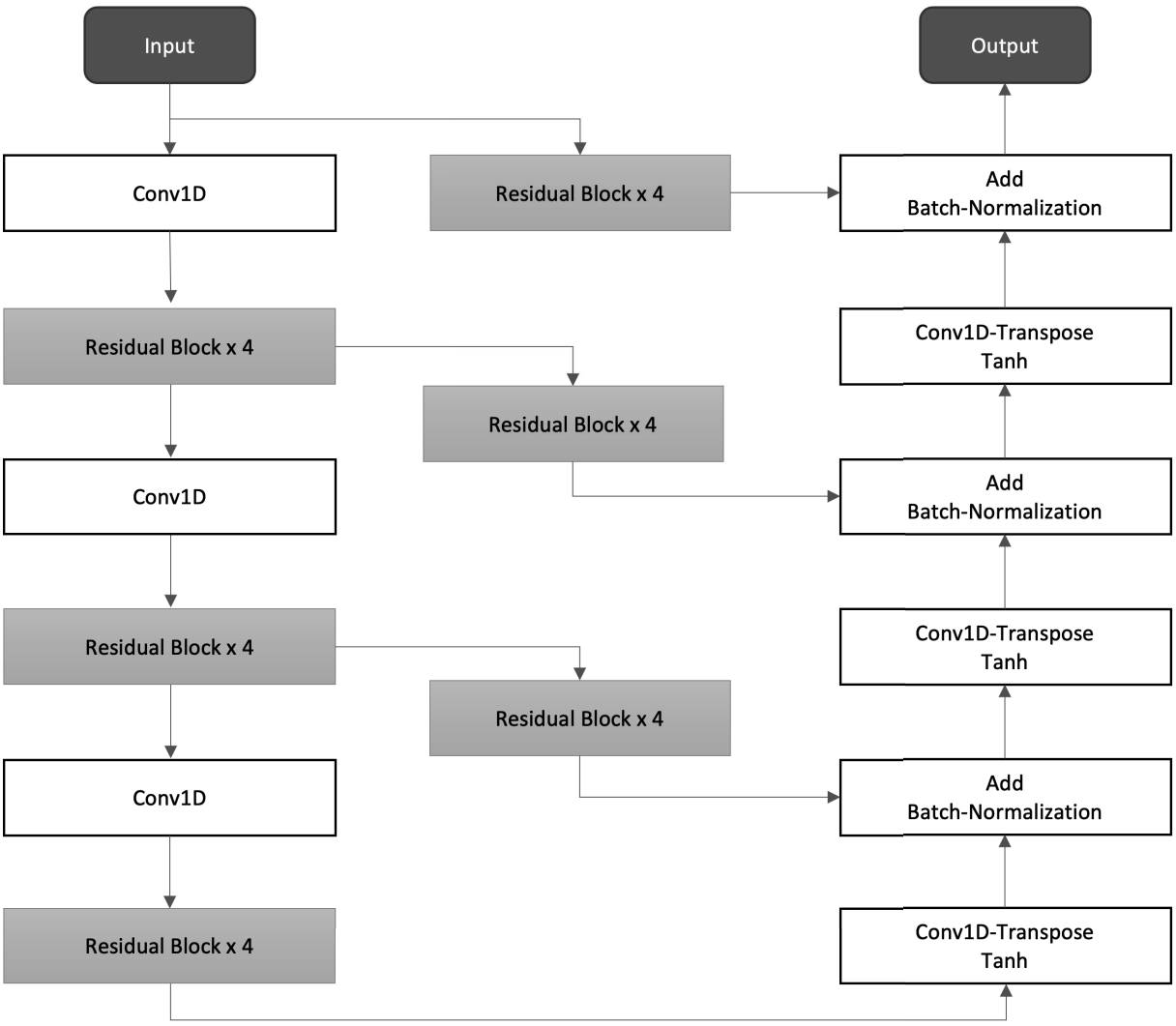


Figure 5.4: U-Net Block: The Residual Block in U-Net is shown in Figure 5.6. Each Residual Block in this U-Net is stacked four. One-dimensional convolution is used to obtain highly abstract features. The parallel bypass aims to learn while maintaining detailed information. For up-sampling, the transpose of One-dimensional convolution is used.

U-Net

The U-Net structure used by Generator and Estimator is shown in Figure 5.4. The Residual Block in this U-Net is shown in Figure 5.6. Each Residual Block in this U-Net is stacked four. One-dimensional convolution is used to obtain highly abstract features. The parallel bypass aims to learn while maintaining detailed information. For up-sampling, the transpose of One-dimensional convolution is used.

Residual Block

The structure of the Residual block is shown in Figure 5.6. In the Residual Block, a single block comprises three layers of 1D convolution, tanh as the activation function, and batch-normalization. The output of the residual block is the sum of the output of these two blocks stacked and the first input. As a result, the residuals from the inputs are learned. The activation function tanh is empirically known to be suitable for learning GANs. Using Batch Normalization also makes the learning process faster and more stable without Dropout.

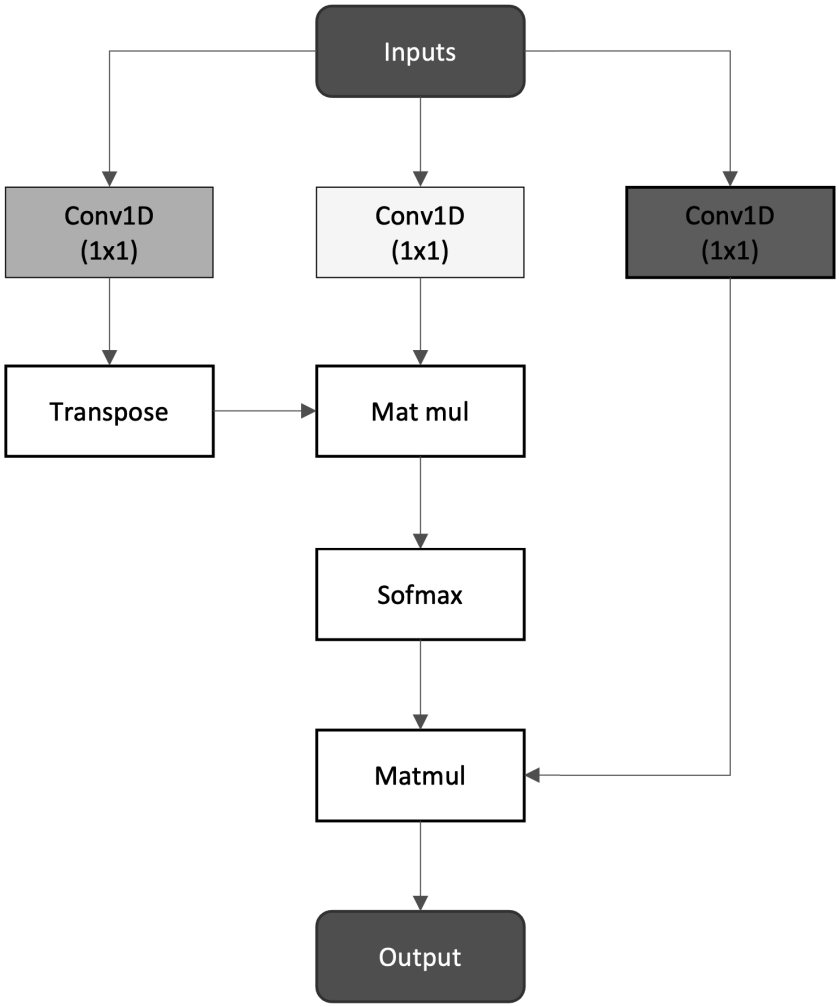


Figure 5.5: Self-Attention Block: The procedure for self-attention is to first find the inner product for a tensor, the output of the yellow layer, with the transpose of that tensor, the output of the green layer. The result of this inner product that applied softmax as an activation function is called an attention map. Then, the attention map, as a weight to be gazed at, is multiplied by the input tensor, the output of the purple layer.

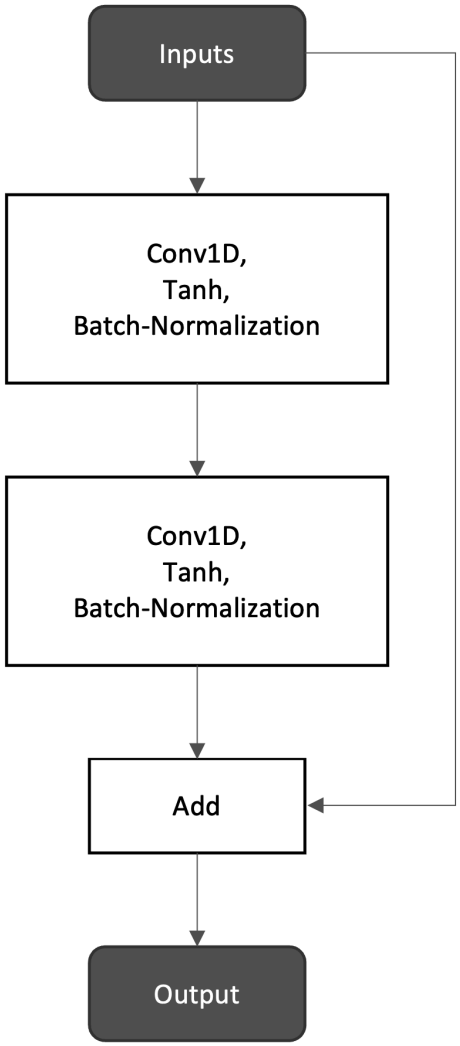


Figure 5.6: Residual Block: In the Residual Block, a single block comprises three layers of 1D convolution, tanh as the activation function, and batch-normalization. The output of the residual block is the sum of the output of these two blocks stacked and the first input. As a result, the residuals from the inputs are learned. The activation function tanh is empirically known to be suitable for learning GANs. Using Batch Normalization also makes the learning process faster and more stable without Dropout.

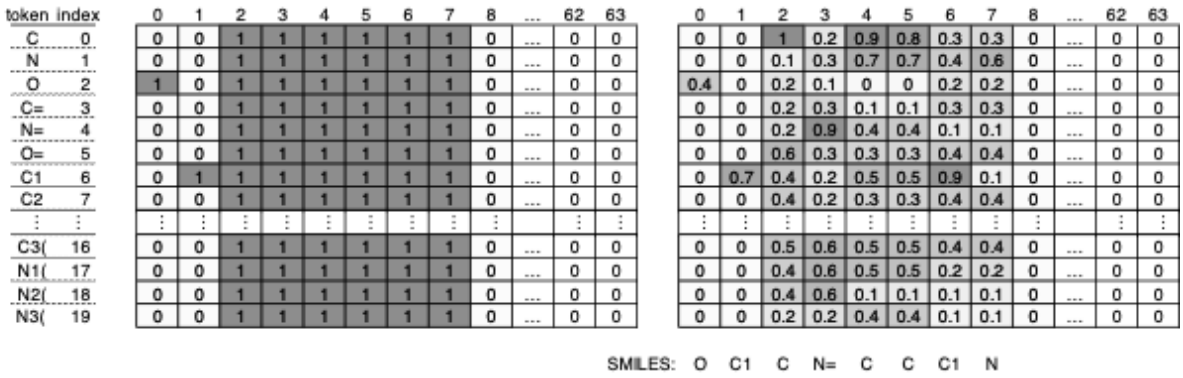


Figure 5.7: Data representation example; the lefthand is the input example, and the lefthand is the output one. Each row represents the index of the token list; each column represents the order of atoms in the generated molecule.
The degree of freedom explained in section 5.3.4 is 38 (=2by19) in the input example. The number of atoms in a generated molecule is randomly generated in the input. In this example, the number of atoms is eight. The first two atoms are randomly generated, and the rest are not yet determined as input. The output is multiplied by the input at the last layer in the generator.

5.2.2 Data Representation

The following list of SMILES [42] elements has been used.

```
['C', 'N', 'O', 'C=', 'N=', 'O=', 'C1', 'C2', 'C3',
 'N1', 'N2', 'N3', 'C(', 'N(', 'C1(', 'C2(', 'C3(',
 'N1(', 'N2(', 'N3(' ]
```

SMILES stands for Simplified molecular-input line-entry system. C, N, O, and F stand for carbon, nitrogen, oxygen, and fluorine, respectively. The character "=" means double bond between atoms. Hydrogen is included in the generated molecules but is not explicitly indicated. Parentheses denote branches, and numbers denote the starting and ending points of the ring. Parentheses are used in pairs, and the number of occurrences of the ring number must be even. However, unclosed parentheses can occur, and the number of occurrences of the ring number can be odd. Therefore, a method to easily obtain a grammatically correct sequence of SMILES tokens has been proposed by O'Boyle N et al.[29] In this study, function F_0 is used to increase the probability of SMILES grammar correctness. In the function F_0, The symbols of parentheses are converted so that they are always paired for parentheses.

A one-hot encoding representation that draws the index of the above list was used. An example of input data for DGN is shown in figure 5.7 lefthand. The righthand in figure 5.7 is an example of generator output, one-hot encoding SMILES.

5.2.3 Training

The estimator E in DGN is used in the training phase. The following pseudo code 3 shows the learning procedure for DGN. For the hyperparameters, the batch size is 8, the number of training iterations is 8192, and using Adam [21] optimizer with a learning rate of 10^{-5} for G Generator, 10^{-4} for D Estimator of Calculation model.

Where, F_0 is a function that takes x, the generated SMILES as input, and returns the SMILES with the grammatically incorrect parentheses and ring numbers corrected as output. F_1 is a function that returns the values of QED, SAS, and $logP$, described in Chapter 4.

The function $logcosh$ at operation14 and 15 in the algorithm 3 is defined in equation (5.1).

$$logcosh(p, y) = \log\left(\frac{e^{p-y} + e^{-p+y}}{2}\right) \tag{5.1}$$

The reason for not using Mean Squared Error(MSE) is that when the error value exceeds 1, the MSE grows as a square, but the $logcosh$ is nearly linear, making the learning process more stable. As the

Algorithm 3 Training Procedure D, G

1: Setting constant values C as the target direction
2: **for** number of iteration **do**
3: **while** mini-batch is not filled **do**
4: z_0, z_1: Randomly generate initial atoms and number-of-atom mask
5: **if** $F_0(G(z_0, z_1))$ is grammatically correct in SMILES **then**,
6: Add the data to the mini-batch.
$z_0 c = z_0, z_1 c = z_1$
7: **end if**
8: **end while**
9: Gathering mini-batch with size m:
10: $\left\{z \mid z_{0c}^{(0)}, \cdots, z_{0c}^{(m)}, z_{1c}^{(0)}, \cdots, z_{1c}^{(m)}\right\}$
11: $\left\{x \mid G(z_{0c}^{(0)}, z_{1c}^{(0)}), \cdots, G(z_c^{(m)} z_{1c}^{(m)})\right\}$
12: $\left\{r \mid F_1(F_0(x^{(0)})), \cdots, F_1(F_0(x^{(m)}))\right\}$
13: $\left\{p \mid E(x^{(0)}), \cdots, E(x^{(m)})\right\}$
14: Updating D with the mini-batch
$$\nabla_{\theta D}\left[\frac{1}{m}\sum_{k=0}^{m}[logcosh(r,\ p)]\right]$$
15: Updating G with the mini-batch
$$\nabla_{\theta G}\left[\frac{1}{m}\sum_{k=0}^{m}[logcosh(C,\ p)]\right]$$
16: **end for**

Figure 5.8: Training Procedure; DGN for molecular Search: Setting constant values C as the target direction 1. For preparing a mini-batch, randomly generate initial atoms and number-of-atom mask z_0, z_1; operation 4. Generate molecule $G(z_0, z_1)$ and evaluate the molecule $F_0(G(z_0, z_1))$ is grammatically correct or not 5.
If the molecule is grammatically correct, add the molecule to the mini-batch 6 Gathering mini-batch with size m as from operation 10 to 13. z represents randomly generated atoms, which can be the seed of grammatically correct molecule; operation 10. x is generated molecules with $G(z)$; operation 11. Calculate measurements results r with function $F_1(F_0(x))$ from x as input; operation 12. Estimate measurements results p with the estimator model D_1 from x as input; operation 13.
The weights of the estimator D_0 are updated from the errors of r and p; operation 14. The weights of the generator G are updated from the error between C and p; operation 15. At this time, G and D_1 are concatenated, and the weight of D_1 is fixed. This means that only G is updated. The operations from 3 to 15 are iterated during training.

two models cooperate, they gradually approach the desired solution. As the prediction accuracy of the computational results around the search area improves, more favorable molecules will be generated.

5.3 Experiment

As an unsupervised learning method, random generation, GA, and DGN are compared for the generation of molecules. ORGAN and MolGAN are compared as supervised learning methods. In both experiments, hydrogen was not included in the number of atoms. Finally, the effect of degree of freedom(DOF) of the input in DGN is confirmed.

5.3.1 Measurements

The measurements of this experiments are follows:

- **Validity** indicates the percentage of the generated SMILES that returned 0 in the function F_0. In other words, for molecules converted to Rdkit[25] Mol objects, we consider them to be grammatically correct SMILES and show the percentage of them.
- **Diversity** indicates the ratio of the number of molecules remaining after eliminating duplicates to the number of molecules generated.
- **QED** stands for Quantitative Estimate of Druglikeliness [6]. QED uses the following eight descriptors to quantify "drug-likeliness." Molecular Weight (MW), logP, number of hydrogen bond donors (HBDs), number of hydrogen bond acceptors (HBAs), polar surface area (PSA), rotatable bonds (ROTBs), number of aromatic rings (AROMs), and number of structural alerts to avoid as drug (ALERTS).
- **SAS** stands for synthetic accessibility score [10]. SAS rates ease of synthesis on a scale of 1 to 10 based on frequency of occurrence and complexity. We used normalized values from 0 to 1.
- **logP** is the predicted octanol/water partition coefficient and is used for estimating fat solubility [43] [7].
- **Sum** is a sum of normalized QED, normalized SAS, and normalized logP.

For QED, SAS, and logP, the values are normalized to values between 0 and 1. For the standardization, we used the module provided by the authors of MolGAN [9].

5.3.2 Comparison

Random generation and EA were compared with DGN. The number of atoms other than hydrogen ranges from 6 to 20 for the first experiment and from 8 to 32 for the second experiment. The number of atom range was applied to Random generation, EA, and DGN.

Random Generation

For random generation, random numbers with Gaussian distribution were used to generate indices for the list of atoms listed in the data representation above. In each experiment, 10000 molecules were generated in SMILES format for each number of atoms. The number of generated molecules is 150000 and 240000 for each experiment. The procedure is shown in Algorithm 4. The list shown in Data Representation in Chapter 3 was used to generate the molecules. A uniform random number was generated, and the random number was used as the index of the list. After the molecules were converted using the function F_0 in Section 3, we attempted to convert them to Mol objects using Rdkit. For the molecules that could be converted to Mol objects, the function F_1 in Section 3 was used to calculate QED, SAS, LogP, and Sum. For molecules that could not be converted to Mol objects, we recorded that SMILES grammar was inappropriate.

Algorithm 4 Random Generation Procedure

```
1: for Number of atoms range do
2:     for number of iteration do
3:         z: Generate random array with uniform distribution
4:         Calculate F_1(F_0(z))
5:     end for
6: end for
```

Figure 5.9: Random generation procedure: Generate uniform random number z as the atom list index; operation 3. Calculate measurements with $F_1(F_0(z))$; operation 4 Iterate from operation 3 to 4 for the number of iteration 2. Iterate from operation 2 to 5 for the number of atoms range 1.

Evolutionary Algorithm

In this comparison, we used the simplest Evolutionary Algorithm(EA) presented by Bäck. T. [4] in chapter 7. The algorithm is shown in Algorithm 5. In EA, if the SMILES grammar of the generated molecule is correct, the score of the generated molecule is computed with the function $F_1(F_0(x))$; if the grammar is incorrect, $F_1(F_0(x))$ returns 0, and the difference between that value and the desired score c is calculated as fitness. As the configuration of EA, the number of generations is from 10 to 40. The best molecules were obtained in smiles format for each atomic number in each experiment for atomic numbers 6 to 20 and 8 to 32. The number of molecules obtained is 450 and 720 for each experiment. The number of individuals in the population is 64. The crossover probability is 0.9. And the probability of an individual mutating is 0.1. As targets, normalized QED=0.8, normalized SAS=0.9, and normalized logP=1.0. The measurements of the molecules generated in each step and the mean squared error (MSE) of the target were calculated. Here, the score was 0 for individuals not grammatically correct in SMILES. Based on the MSE, the individual with the best fit was allowed to survive as the next generation.

The function mse at operation 6 and 10 in the algorithm 5 is defined by the following equation, which is given in Chapter 3 and restated below. (5.2).

$$mse(p, y) = \frac{1}{n}\sum_{i=1}^{n}(p_i - y_i)^2 \tag{5.2}$$

DGN

As a verification phase of the DGN, a randomly generated input was given to Generator, G. The number of initial atoms as input was set to 1 in Experiment 1 with 6 to 20 atoms and 2 in Experiment 2 with 8 to 32 atoms. In each experiment, 100 molecules were generated in SMILES format for each number of atoms. The number of molecules obtained is 1500 and 2400 for each experiment. Like the training phase, the list shown in Data Representation in Chapter 3 was used to generate the molecules. If the generated molecule was parsable, QED, SAS, LogP, and Sum were calculated with $F_1(F_0(G(z_0, z_1)))$ as shown in Algorithm 6

5.3.3 Result

A comparison of the molecules generated by unsupervised learning (Random, EA, DGN) and supervised learning (MolGAN, ORGAN) is shown in Table 5.1. The QED, SAS, and logP in the table are mean of the generated molecules values. ORGAN, MolGAN, and MolGAN(QM9) results were obtained from the Objective ALL row of Table 2 from the original paper [9]. QED corresponds to Druglikeliness, SAS to Synthesizability, and logP to Solubility. Sum in the table is the sum of QED mean, SAS mean, and logP mean.

Algorithm 5 Evolutionary Algorithm Procedure

1: **for** Number of atoms range **do**
2: **for** number of population **do**
3: Generate x as individuals with uniform distribution for initial $population$
4: **end for**
5: evaluate $population$
6: $fitness = mse\,(F_1(F_0(x)), c)$
7: **for** number of generation **do**
8: Apply crossover and mutation on the $population$
9: evaluate $population$
10: $fitness = mse\,(F_1(F_0(x)), c)$
11: Select individuals from $population$ based on the $fitness$ as next generation
12: **end for**
13: Get individual with the best result x_{best}
14: Calculate $F_1(F_0(x_{best}))$
15: **end for**

Figure 5.10: Evolutionary Algorithm Procedure: Initialize population; operation 2 3 Calculate measurements $F_1(F_0(x))$ of individual molecule x and $fitness$ with mse; operation 4. Apply crossover and mutation on the $population$; operation 8 Calculate measurements $F_1(F_0(x))$ of individual molecule x and $fitness$ with mse as evaluation; operation 10 Select individuals from $population$ based on the $fitness$ as next generation; operation 11 Iterate from 8 to 11 for number of generations; operation 7 After the iteration, Get the individual with the best result x_{best}; operation 13. Calculate the measurements of the best $F_1(F_0(x_{best}))$; operation 14. Iterate from 2 to 14 for the number of atoms-range; operation 1

Algorithm 6 DGN Validation Procedure

1: **for** Number of atoms range **do**
2: **for** number of iteration **do**
3: z_0, z_1: Randomly generate initial atoms and number-of-atom mask
4: **if** $F_0(G(z_0, z_1))$ is parsable **then**
5: Calculate $F_1(F_0(G(z_0, z_1)))$
6: **end if**
7: **end for**
8: **end for**

Figure 5.11: DGN Validation Procedure: Randomly generate initial atoms and number-of-atom mask, z_0, z_1; operation 3. Generate molecule $G(z_0, z_1)$ and confirm whether the molecule is parsable; operation 4. If it is parsable, it means grammatically correct, calculate measurements with $F_1(F_0(G(z_0, z_1)))$; operation 5. Iterate from operation 3 to 6 for the number of iteration 2. Iterate from operation 2 to 7 for the number of atoms range 1.

Molecules that were grammatically correct (can be analyzed with Rdkit) and generated molecules that eliminated SMILES duplications were included in the calculations. The results for MolGAN and ORGAN are taken from the original MolGAN paper [9], where QED corresponds to Druglikeliness, SAS to Synthesizability, and logP to Solubility.

Algorithm	Sample	Validity(%)	Uniqueness	QED	SAS	logP	Sum
Random(6-20)	150000	21.8	0.99	0.36	0.05	0.17	0.53
Random(8-32)	240000	10.8	1.00	0.35	0.02	0.09	0.46
EA(6-20)	450	100.0	0.90	0.59	0.69	0.75	2.02
EA(8-32)	720	100.0	0.98	0.62	0.46	0.71	1.79
DGN(6-20)	1500	86.8	0.11	0.51	0.70	0.82	**2.03**
DGN(8-32)	2400	96.7	0.15	0.46	0.56	0.87	**1.89**
ORGAN	6400^{*1}	96.1	0.89^{*2}	0.52	0.71	0.53	1.76^{*3}
MolGAN	6400^{*1}	97.4	0.02^{*2}	0.47	0.84	0.65	1.96^{*3}
MolGAN(QM9)	6400^{*1}	98.0	0.02^{*2}	0.51	0.82	0.69	2.02^{*3}
QM9	133247^{*4}	100.0	1.00	0.46	0.22	0.28	0.96

Table 5.1: The results for MolGAN and ORGAN are taken from the original MolGAN paper [9], where QED corresponds to Druglikeliness, SAS to Synthesizability, and logP to Solubility. $*1$: based on MolGAN paper [9]. $*2$: Uniquness were multipleied with Unique(%) and Diversity as meaningful ratio. $*3$: Sum is sum of QED(Druglikeliness), SAS(Synthesizability), and logP(Solubility). $*4$: The original QM9 dataset contains 133885 molecules. Easily convertible to RdKit mol objects were 133247.

For each unsupervised learning method, the distribution of the score, the sum of QED, SAS, and logP at each number of atoms is shown in Figure 5.12 as a kernel density estimate. EA and DGN scores are higher than randomly generated ones for all atomic numbers. The scores of EA and DGN are close. However, for EA, the scores tend to decrease as the number of atoms increases, as shown in the results for atom numbers 8 32 in Figure 5.12. The increase in the number of atoms means an increase in the number of combinatorial patterns and the complexity of the problem. The increased complexity makes it difficult for a random choice algorithm such as EA to obtain good scores for individuals with mutation and crossover in the next generation. DGN, on the other hand, can obtain relatively high scores even as the number of atoms increases. This is because DGN is trained in the search space by gradient descent, thus avoiding combinatorial explosion and performance degradation.

Images of generated molecules are show in Figure 5.13, 5.14, 5.15 and 5.16. Randomly generated molecules have a higher percentage of complexity. For EA, molecules become more complex as the number of atoms increases. For DGN, many are relatively simple even as the number of atoms increases.

5.3.4 Degree of freedom of DGN input

The Degree of Freedom (DOF) refers to the variety of input arrays of the generator model in DGN. DGN(DOF=38(=2by19)) and DGNr(DOF=1216(=64by19)) are compared. As a result, in table 5.1 and score distribution in Figure 5.18, the scores of DGNr are much lower than DGN. It is almost the same as random generation. However, Div: diversity of DGNr is greater than DGN. During training, loss values of DGNr are greater than DGN in figure 5.17. It clearly shows that DGNr needs to be better trained.

According to these results, DOF affects the convergence ability of training and the diversity of products. It is a trade-off. You might have to determine DOF depending on the complexity of the problem to solve.

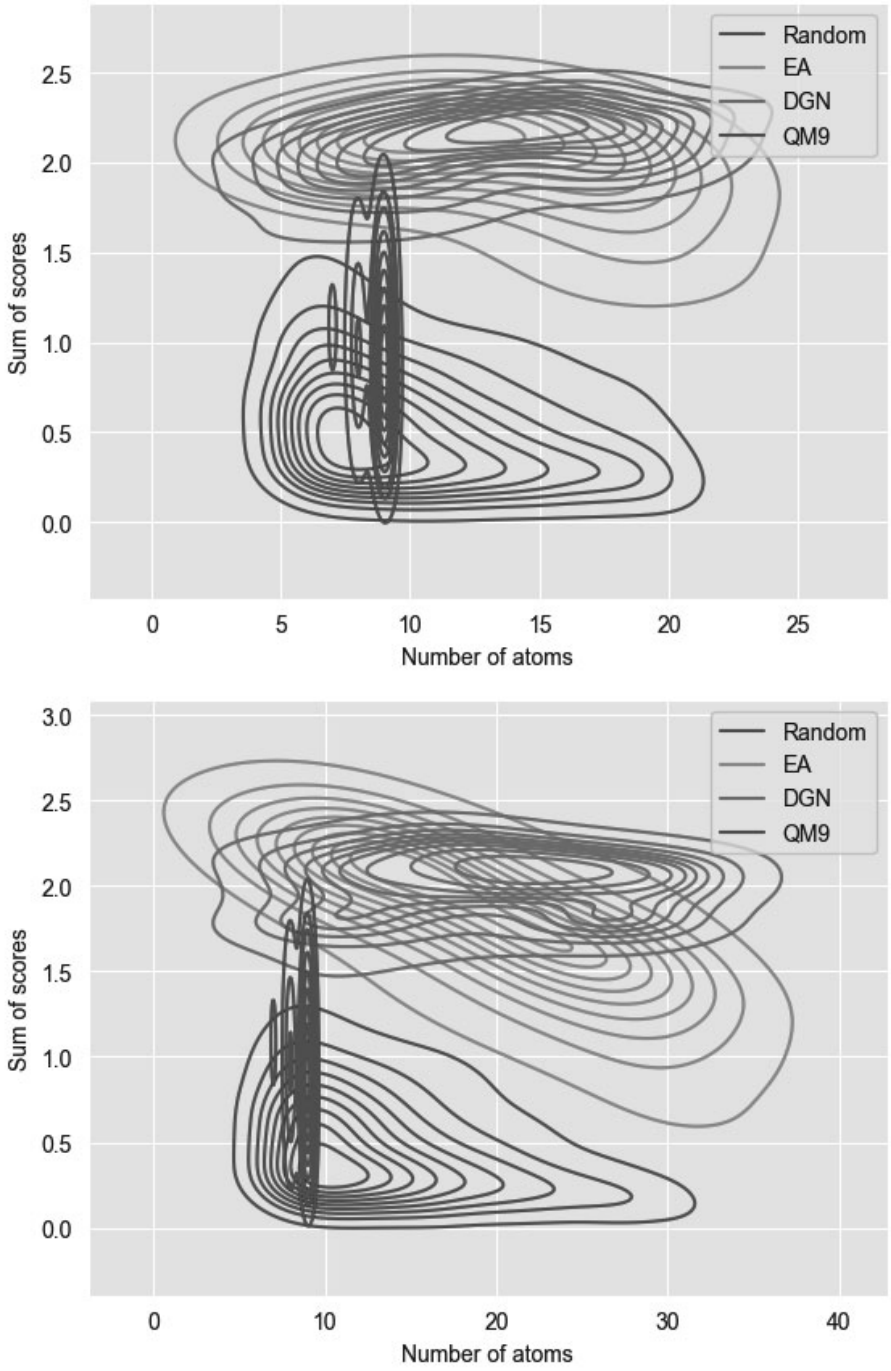


Figure 5.12: On the top is Experiment 1, the number of molecules is 6 to 20. On the bottom is Experiment 2; the number of molecules is 8 to 32. The kernel density estimates of the score distribution according to the number of molecules. The x-axis is the number of atoms in the generated molecules. The y-axis is the sum of QED, SAS, and logP. Blue is random generation, orange is EA, and green is DGN. Red is not generated but is a direct calculation of the original QM9 dataset for reference. Random scores were lower than the others. As the number of atoms increases, the score decreases. EA scores generally well, but performance declines as the number of atoms increases. DGN performs well, and performance does not decrease as the number of atoms increases.

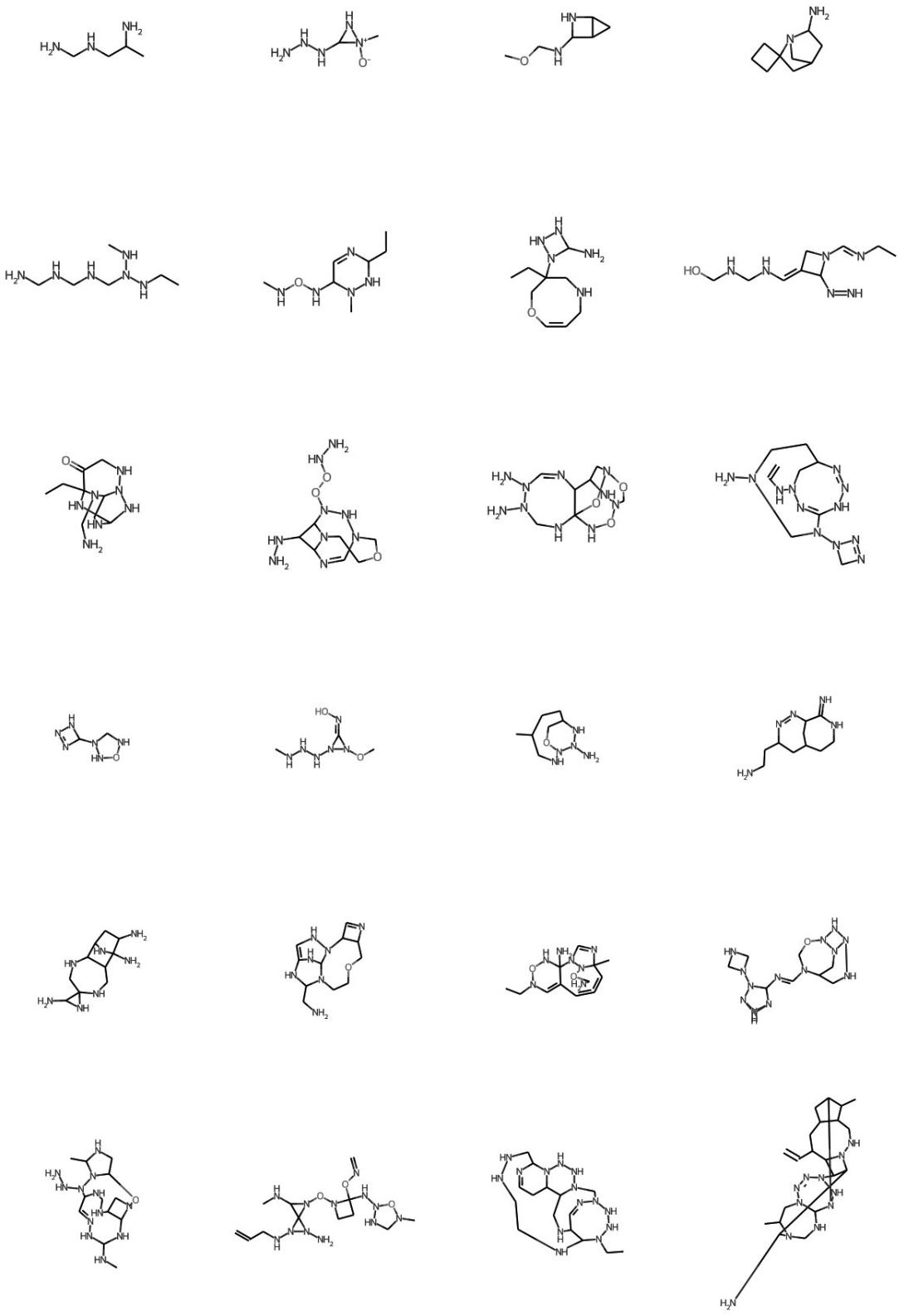


Figure 5.13: Molecular examples that are randomly generated. Rows one through four are Experiment 1 (number of atoms excluding hydrogen, 6-20). Rows five through eight are Experiment 2 (number of atoms excluding hydrogen, 8-32). Many molecules have complex shapes.

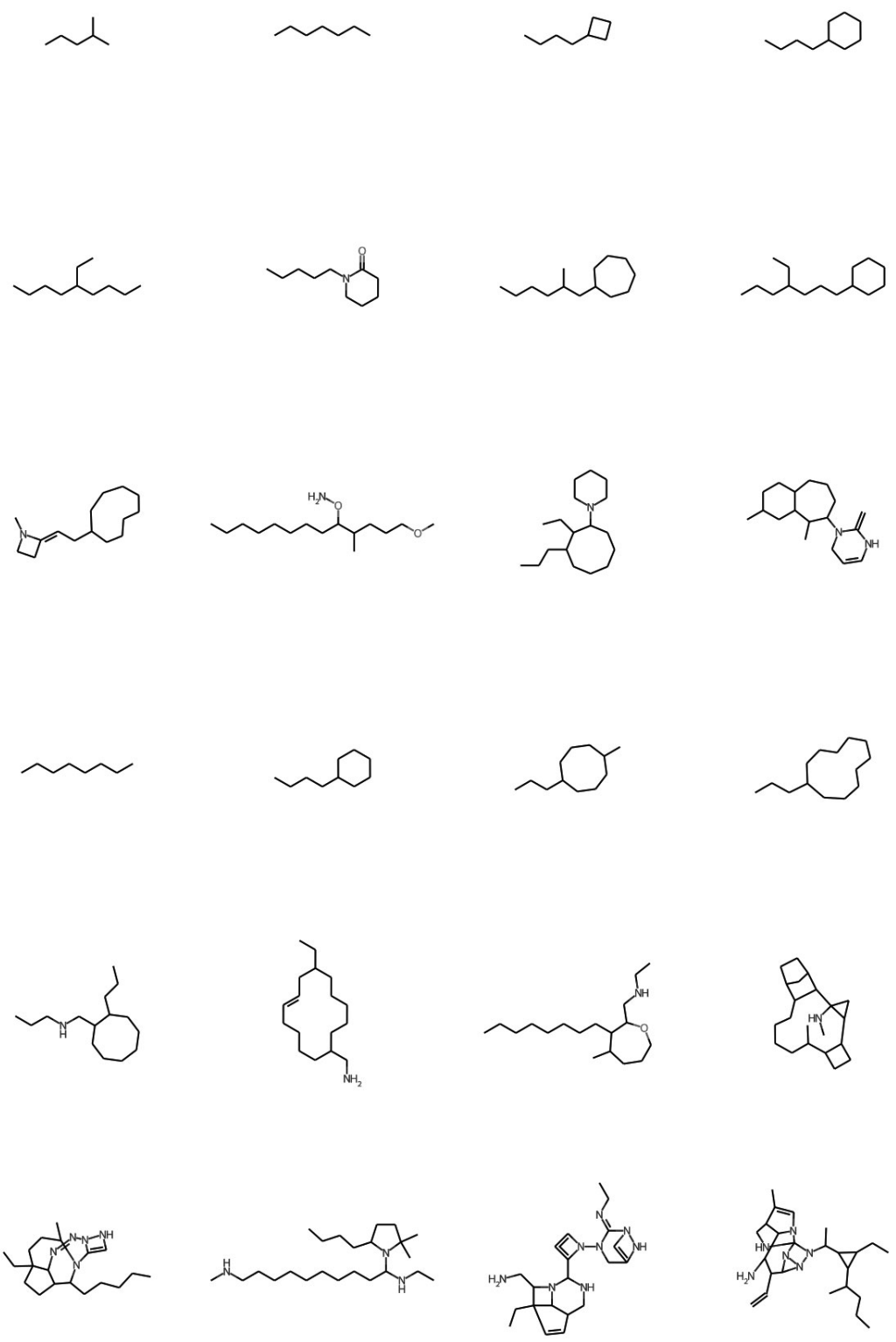


Figure 5.14: In molecular examples generated by EA, the duplicates are eliminated. Rows one through four are Experiment 1 (number of atoms excluding hydrogen, 6-20). Rows five through eight are Experiment 2 (number of atoms excluding hydrogen, 8-32). Molecules with a small number of atoms are often simple. Molecules with more than 20 atoms have more complex shapes.

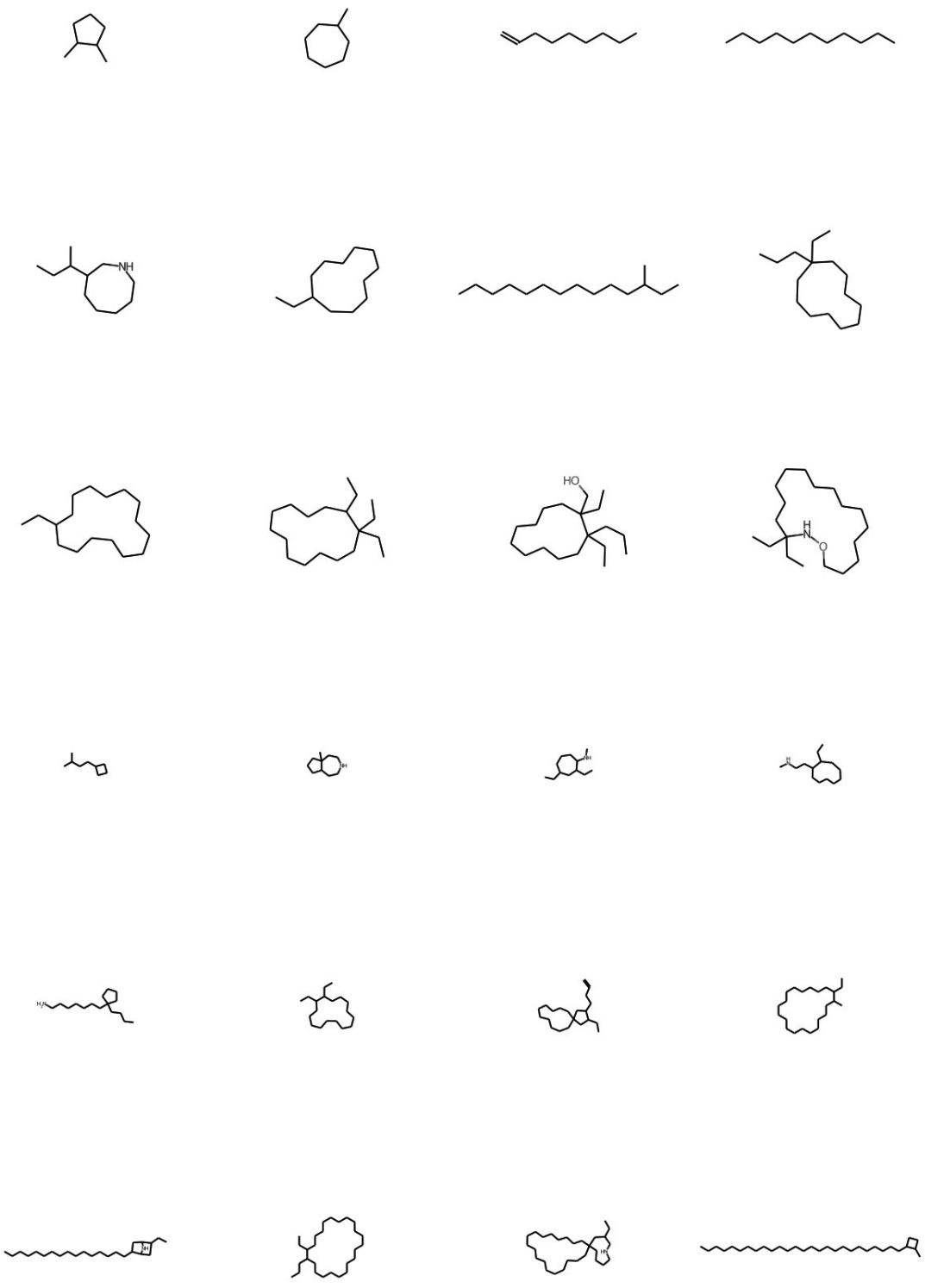


Figure 5.15: In molecular examples generated by DGN, the duplicates are eliminated. Rows one through four are Experiment 1 (number of atoms excluding hydrogen, 6-20). Rows five through eight are Experiment 2 (number of atoms excluding hydrogen, 8-32). Molecules are often simple. Even when the number of atoms exceeds 20, they maintain their simple shapes.

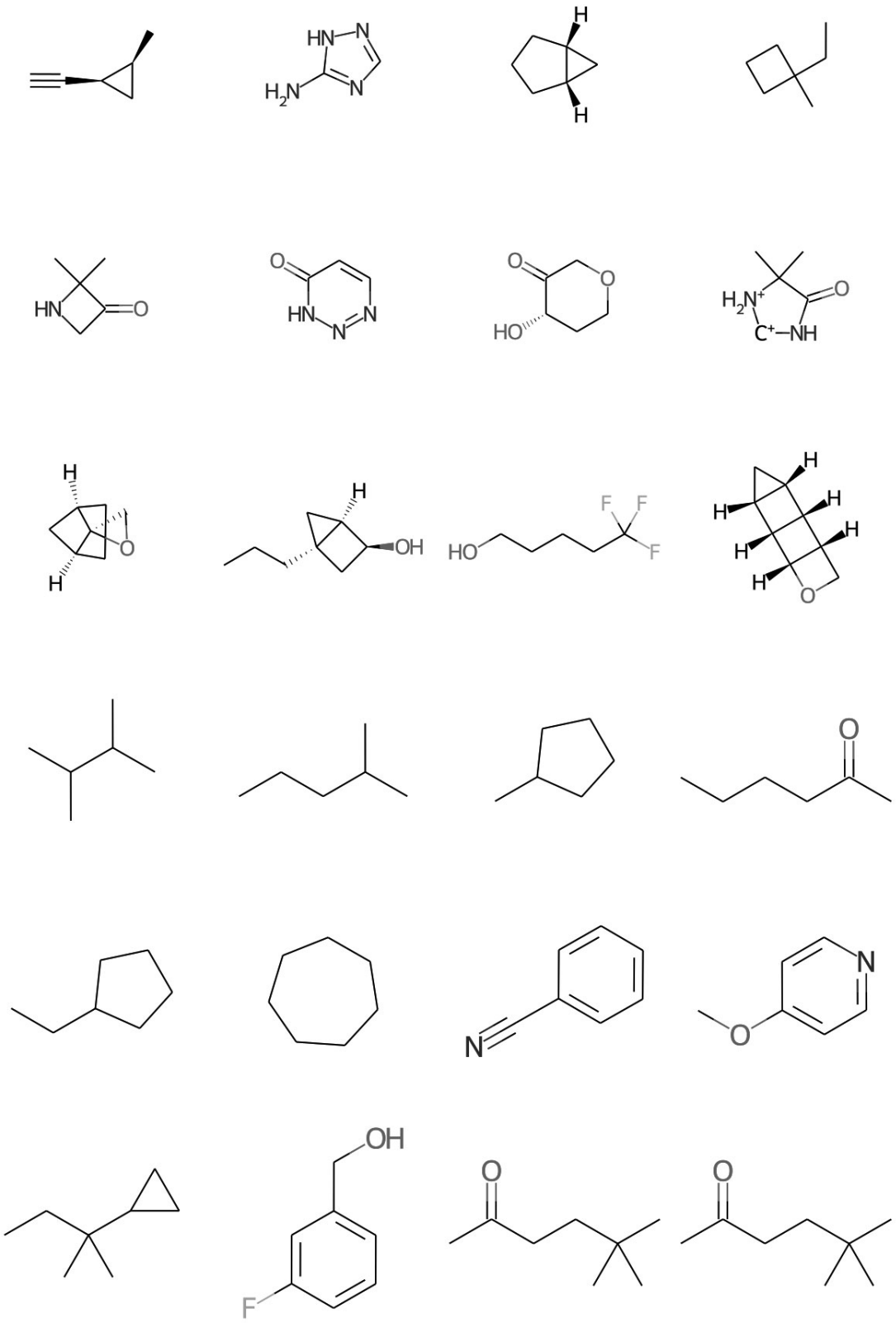


Figure 5.16: Molecule examples included in the QM9 dataset. In the first to the fourth row, the number of atoms in the molecule is 6 to 9, excluding hydrogen. In the fifth to eighth row, the number is also from 6 to 9, excluding hydrogen, and the sum of normalized scores is over 1.9. The molecules in rows five through eight are simpler than the others.

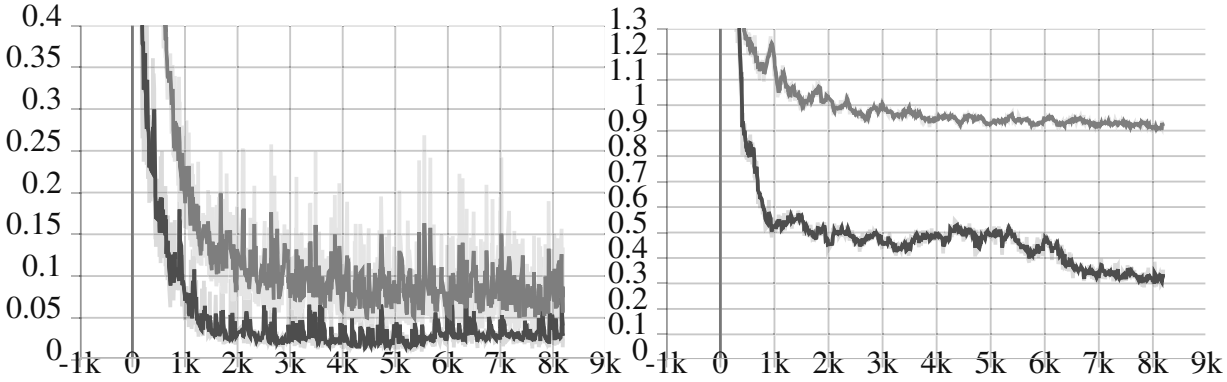


Figure 5.17: DOF Comparison of time-history of loss values during training; The left-hand is the loss value transition of the estimation model. The righthand is the loss value transition of the combined model. The blue line is DGN(DOF=38), the orange one is DGNr(DOF=1216(=64by19)). DGNr is difficult to converge.

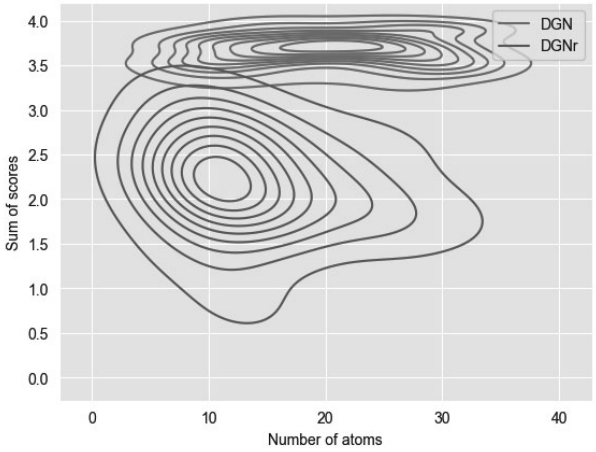


Figure 5.18: Comparison to Sum6 score between DGN and DGNr

5.4 Summary of the Chapter

The results of this study suggest that DGN can generate molecules without training data and with features comparable to models from supervised learning. In general, the dataset used for training is smaller than the search space, and the features obtained by the dataset do not necessarily encompass the desired solution. Suppose the function that evaluates the generated object, in this study, the evaluation of the molecule by F_0 and F_1, satisfies the necessary and sufficient conditions. In that case, the DGN can function as an excellent generator. A similar method is EA, but EA's performance degrades as the complexity of the problem increases due to combinatorial explosion. On the other hand, DGN can asymptotically approach the solution by gradient descent even when the problem is complex and thus is more likely to produce a good solution than EA. Since preparing datasets for training is still expensive, DGNs can complement conventional supervised learning.

As a disadvantage of DGN, the evaluation function, e.g., $F_1(F_0(x))$ in this study, is often simpler than in reality. If the evaluation function satisfies the necessary conditions but not the sufficient conditions, the output of the DGN may be biased. Therefore, when using DGN, appropriate constraints based on domain knowledge should be applied to input data and models.

For future research, it is necessary to devise a data representation less prone to stagnation near the local optimum. Increasing the input degrees of freedom is necessary to obtain various solutions in GAN-like models. However, high input degrees of freedom have the side effect of making learning convergence more difficult. By considering data representations that can maintain appropriate outputs, such as grammatical correctness, we expect to achieve diverse outputs and learning convergence, even with various inputs.

Chapter 6

Conclusion

6.1 Findings

6.1.1 Applicability of DGN

As described in Chapter 1, this study focused on the structure of products in the industrial sector. Designers and engineers often use several tools to evaluate product designs to select the most appropriate design among various candidates. Their evaluation tools are based on the structure of the product. This study proposed a DGN incorporating that tool as an evaluation function in a deep learning model. DGN is applied to building structure design and molecular search and confirmed its effectiveness.

Structural design

In building structural design, the model was applied to the shear panel's placement in buildings and was found to be appropriately learned and practical. In addition, the learned generative model acquired a certain generalization capability. The eccentricities and DCRs calculated from the shear panel arrangements generated from the test data were generally adequate, although there were some outliers.

The loss function values of the estimator model, D_0, are well converged. The loss function values of the concatenated model $D_1(G)$ also converge well. It was confirmed that even regression models could be learned without divergence, rather than GAN-like models that are true-or-false classifiers.

Eccentricity and DCR were used as indices to evaluate the shear panel arrangement. The eccentricities and DCRs calculated from the shear panel configurations generated from the test data were generally adequate, although there were some outliers.

When the DGNs generated model is used as a design support tool, multiple shear panel placement patterns are generated from the building's architectural data. Of the multiple shear panel arrangements generated, the structure containing anomalies is discarded. The preferred one is selected from the remaining designs. Used in this manner, it is a valuable engineering tool.

Molecular Search

DGN's performance was comparable to that of conventional methods on the task of molecular search. It was compared to ORGAN and MolGAN based on GAN as conventional methods. It was also compared to random generation and EA.

ORGAN and MolGAN combine GAN with reinforcement learning methods that feedback rewards for generated molecules with favorable features. They use drug-likeliness, solubility, and synthesizability as evaluation indices. DGN met results comparable to ORGAN and MolGAN for those evaluation metrics, even though it did not use training data.

As for unsupervised learning, we compared DGN with random generation and EA. Unsurprisingly, the simple random generation did not perform well. For EA, the performance was excellent for molecules with a small number of atoms. However, performance degraded as the number of atoms increased. DGN

showed almost no performance degradation despite the increased number of atoms. The superiority of DGN for complex problems was confirmed.

6.1.2 Advantage of DGN over EA

DGN can serve as a good generator if the function that evaluates the generated object, the evaluation function, F_0, and F_1 in Chapter 5, satisfies the necessary and sufficient conditions. A similar method is EA, but EA's performance degrades due to combinatorial explosion as the complexity of the problem increases. On the other hand, DGN is likely to generate better solutions than EA because it can approach the solution asymptotically and by gradient descent, even if the problem is complex.

6.1.3 Elimination of bias derived from Data

Regarding data bias, DGN may complement traditional supervised learning. In general, training data contain bias. The DGNs in Chapters 4 and 5 were able to train without using pre-prepared correct data.

In general, the dataset used for learning is smaller than the search space, and the features obtained by the dataset do not necessarily encompass the desired solution. It is challenging to remove the bias of the data used for learning concerning the entire search space. To remove the bias, a dataset that encompasses the search space should be prepared. However, the preparation of the data set is enormously expensive.

In the structural design in Chapter 4, we used building data as constraints for placing shear panels in the DGN training. Still, we did not use the shear panel placement data as ground truth corresponding to that building. In Chapter 5, only the list of atoms that make up the molecule was provided, and no data was used in the training. The results in Chapter 5 suggest that DGN can generate molecules without training data and with features comparable to models based on supervised learning.

Although DGN cannot explore all of the search space due to computational resource constraints, it may complement conventional supervised learning concerning data bias.

6.2 Future works

Inputs' Degrees of Freedom

Through the try-and-error of the hyper-parameter tuning for DGN training, we found that the degrees of freedom(DOF) of the input of the generator G affect the convergence of the learning and the diversity of the products.

As a matter of course, if the input of the DGN's generator G is constant values, and the target, C, is constant values, the weights of the generator and guesser will converge to a single state according to the initial values of the each layers' weights. Conversely, output diversity can be expected with a high DOF input of the generator G of the DGN. At the same time, convergence becomes problematic if the input with a high DOF. In other words, convergence and diversity are in a trade-off relationship, with the degree of freedom of the input as a parameter.

What DOF of input is required depends on the design target and customer requirements. Therefore, it is difficult to say how many DOF are generally good. However, moderate DOF restrictions are believed to affect the usefulness of the DGN as a design assistance tool. This study has yet to examine thoroughly how many DOF are good. This is a subject for future research.

Limitation of evaluation functions

One drawback of DGN is that the evaluation function, e.g., $F_1(F_0(x))$ in Chapter 5, is often simpler than in reality. Even if the evaluation function satisfies the necessary conditions, the output of the DGN may be biased if it does not meet sufficient conditions. Therefore, when using DGNs, applying appropriate constraints based on domain knowledge put data and model structure is desirable.

Other Applications

It is worthwhile to explore the applicability of DGN to other design-like tasks similar to structural design and molecular search. DGN can potentially be applied to tasks for which conventional generative methods cannot be used. DGN is expected to improve the productivity of design tasks. DGN can be used for learning if the following conditions are met.

- If the evaluation function can be incorporated into the model as a function F, although it is not easy to rewrite as a differentiable function. For example, in the case of LSI design, tools for evaluating device layout optimization, wiring optimization, etc., can be used as DGN function F. In the case of mechanical element design, finite element analysis(FEA) tools used for stress analysis, temperature analysis, etc., can be used as DGN function F.
- If the problem is not simple enough to be solved by EA/GA.
- When it is possible to express appropriate data about the design target and design constraints.

For simple problems, GA and other tools have been used so far. DGNs can be applied to more complex problems. One advantage of DGN is that these functions do not need to be rewritten as differentiable functions. Therefore, it is believed that productivity can be improved without significantly changing conventional business procedures simply by devising data representation. It means DGNs can be applied to various industries.

Combination with EA or GP

Since DGN and EA employ different search strategies, combining them may have complementary effects. DGN, an application of deep learning, uses the gradient descent method, while the basic strategy of EA is random selection. By taking advantage of the characteristics of each method, better performance can be obtained by combining wide-area search using random selection and narrow-area search using the gradient descent method.

When the search space is vast, the random selection algorithm has difficulty converging, and the gradient descent method tends to stay at the local optimum. In such cases, it is compressing the search space by some means. If a graph representation can be adopted from the nature of the problem, it is possible to reduce the search space effectively.

For EA, genetic programming(GP) can be used to handle graph representation. In deep learning, graph convolutional networks can be used. GP is a method that represents individuals as a graph structure and randomly recombines branches of the graph to obtain a better solution. It can be applied to any object a graph represents, such as mathematical expressions, sentences, and molecules. The graph structure restricts data representation, reducing the search space. Therefore, if a suitable representation format can be defined, better performance can be obtained compared to the usual genetic algorithm or DCN, and further performance improvement can be expected by combining DGN and GP using graph representation.

Chapter 7

Appendix

7.1 Structural Design Method Based on Japanese Standard Act

As described in Chapter 3, the design of architectural or civil engineering structures assumes various external forces and makes sure that the design meets the criteria for those external forces. This chapter describes a method in the Japanese Building Standard Act for considering the layout of shear panels with seismic forces as the external force. The Japanese Building Standard Act provides a two-stage design method: one is to design the structure for rare seismic forces in a generally elastic range (primary design), and the other is to allow plastic deformation of the structure for extremely rare earthquakes and to design for the energy-absorbing capacity of the framework (secondary design). This section describes the primary design in detail. This explanation is based on the explanation of the structure-related provisions of the Japanese Building Standard Act [24].

External Force

External forces acting on the structure are assumed to ensure that it is safe during its service life. The building structure system's response to those external forces is verified to meet the criteria. Buildings constructed in Japan are assumed to be subjected to long-term loads such as dead weight and loading and short-term loads such as earthquakes, wind, and snow. This section describes the calculation of seismic loads, often the dominant design loads.

Here, we consider seismic motions that rarely occur during the service life of the building. The horizontal force at the lowest level, which varies moment by moment with the time of the seismic motion, is substituted for a static load of 20% of the weight of the building. Its ratio of 20% to the building weight is called the standard story shear force coefficient, expressed as $C_0 = 0.2$.

Ai distribution

A simplified method for calculating the natural period T of a building is as follows 7.1.

$$T = h(0.02 + 0.01a) \tag{7.1}$$

The following equation 7.2 expresses the distribution of horizontal forces in the height direction of the building, the so-called Ai distribution, taking into account the effects of higher-order modes.

$$A_i = 1 + \left(\frac{1}{\sqrt{a_i}} - a_i \right) \frac{2T}{1 + 3T} \tag{7.2}$$

Demand-Capacity Ratio

The demand-capacity ratio (DCR) is a measure used to assess the safety of a structure under different loading conditions. It is defined as the ratio of the maximum expected demand on a structure to its capacity to resist that demand.

In other words, the DCR indicates how close a structure is to its maximum capacity when subjected to a particular load or combination of loads. A DCR greater than 1.0 indicates that the demand on the structure is greater than its capacity, which means the structure is potentially unsafe and may be at risk of failure.

For example, in the case of a building, the demand on the structure would be the maximum load that the building is expected to experience, such as the weight of the people and furniture, the wind forces, or seismic forces. The structure's capacity would be its ability to resist these loads, which is determined by the strength and stiffness of the building components, such as the columns, beams, and shear panels. If the demand on the building exceeds its capacity, the DCR will be greater than 1.0, indicating that the structure is potentially unsafe and may need to be strengthened or repaired to reduce the risk of failure.

Engineers use DCR as a critical factor in assessing the safety of a structure and determining whether it needs to be repaired, strengthened, or replaced to ensure that it can safely withstand the loads it is expected to experience throughout its design life.

Eccentricity

Floor slabs generally have higher in-plane stiffness than walls and columns, in building structures. Therefore, technically, the floor slab is generally regarded as a rigid body and designed as if seismic forces were acting on the center of gravity of the floor slab. Eccentricity is the ratio of the center of gravity to the eccentricity of the rigid body as torsional resistance. The higher the value of eccentricity, the greater the effect of eccentricity. The value of eccentricity can be obtained by the following procedure.

The eccentric distance e_x, e_y between the weight center and the stifness center of a the structure. gravity center is defined following equation 7.3 7.4.

$$g_X = \frac{\sum(N \cdot X)}{W} \tag{7.3}$$

$$g_Y = \frac{\sum(N \cdot Y)}{W} \tag{7.4}$$

Where W is a sum of N, the vertical force of each column 7.5.

$$W = \sum N \tag{7.5}$$

Stiffness center of the structure is obtained from followings 7.6 7.7. The stiffness in each direction is the sum of the effective seismic elements in the X- and Y-directions, respectively.

$$l_X = \frac{\sum(K_Y \cdot X)}{\sum K_Y} \tag{7.6}$$

$$l_Y = \frac{\sum(K_X \cdot Y)}{\sum K_X} \tag{7.7}$$

The eccentricity distance e_X, e_Y is calculated from the position of the center of gravity g_X, g_Y and stiffness center l_X, L_Y as follows 7.8 7.9.

$$e_X = |l_X - g_X| \tag{7.8}$$

$$e_Y = |l_Y - g_Y| \tag{7.9}$$

Next, the torsional stiffness K_R around the stiffness center is calculated 7.12. The coordinates are moved in parallel with the stiffness center position as the origin.

$$\bar{X} = X - l_X \tag{7.10}$$

$$\bar{Y} = Y - l_Y \tag{7.11}$$

$$K_R = \sum \left(K_X \cdot \bar{Y}^2 \right) + \sum \left(K_Y \cdot \bar{X}^2 \right) \tag{7.12}$$

Therefore, the elasticity radius r_{eX}, r_{eY} is given by the following equation 7.14 7.16.

$$r_{eX} = \sqrt{\frac{K_R}{\sum K_X}} \tag{7.13}$$

$$= \sqrt{\frac{\sum \left(K_X \cdot \bar{Y}^2 \right) + \sum \left(K_Y \cdot \bar{X}^2 \right)}{\sum K_X}} \tag{7.14}$$

$$r_{eY} = \sqrt{\frac{K_R}{\sum K_Y}} \tag{7.15}$$

$$= \sqrt{\frac{\sum \left(K_X \cdot \bar{Y}^2 \right) + \sum \left(K_Y \cdot \bar{X}^2 \right)}{\sum K_Y}} \tag{7.16}$$

Eccentricity R_{ex}, R_{ey} is expressed by the following equation 7.17 7.18 from the eccentricity distance of the structure e_x, e_y and elastic radii r_{ex}, r_{ey}.

$$R_{ex} = \frac{e_y}{r_{ex}} \tag{7.17}$$

$$R_{ey} = \frac{e_x}{r_{ey}} \tag{7.18}$$

7.2 Libraries and Resources

This section lists the main libraries and computational resources used for DGN for Molecukar finding in Chapter 5.

Libraries

Keras [1] was used as the framework for building and training the model. As a backend, TensorFlow [2] is used. Following is a list of the major libraries and Pythons' version.

- keras 2.11.0
- numpy 1.22.3
- python 3.8.16
- rdkit 2022.03.2
- tensorflow 2.11.0

Resources

Following is the major-part list of calculation resources.

- OS: Ubuntu 20.04.6 LTS
- CPU: Intel Core i5-10400 CPU @ 2.90GHz
- Memory: 32GB
- GPU0: Nvidia GeForce GTX 1660 SUPER
- GPU1: Nvidia GeForce GTX 1650 SUPER
- Docker version 23.0.5

7.3 DGN for molecular finding; model details

Generator

Generator model details in DGN for molecular finding are shown in Figure 7.1 7.2 7.3 7.4 7.5. It includes the output shape of each layer.

Estimator

Estimator model details in DGN for molecular finding are shown in Figure 7.6 7.7 7.8 7.9 7.10. It includes the output shape of each layer.

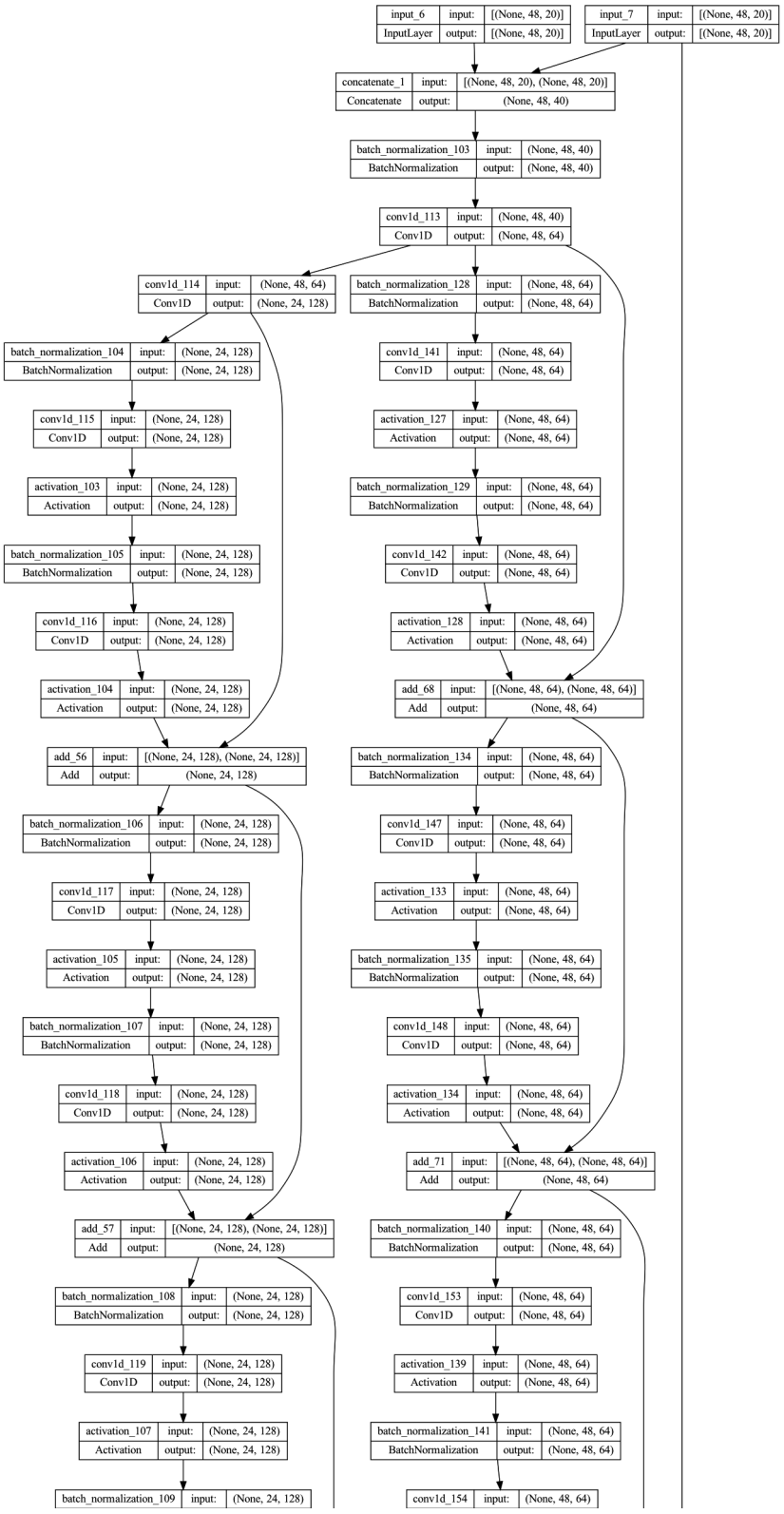


Figure 7.1: DGN for molecular finding; generator 0:

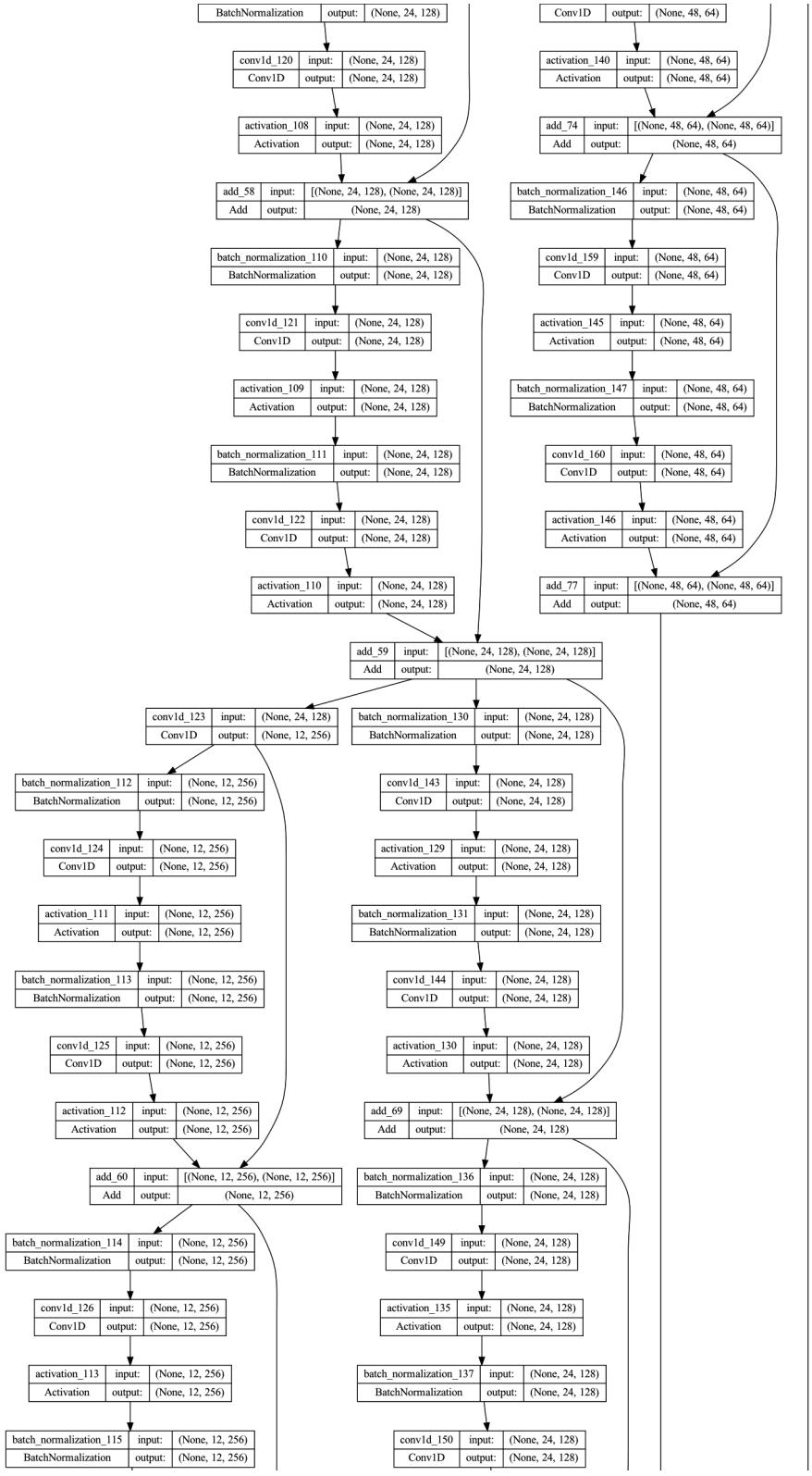


Figure 7.2: DGN for molecular finding; generator 1:

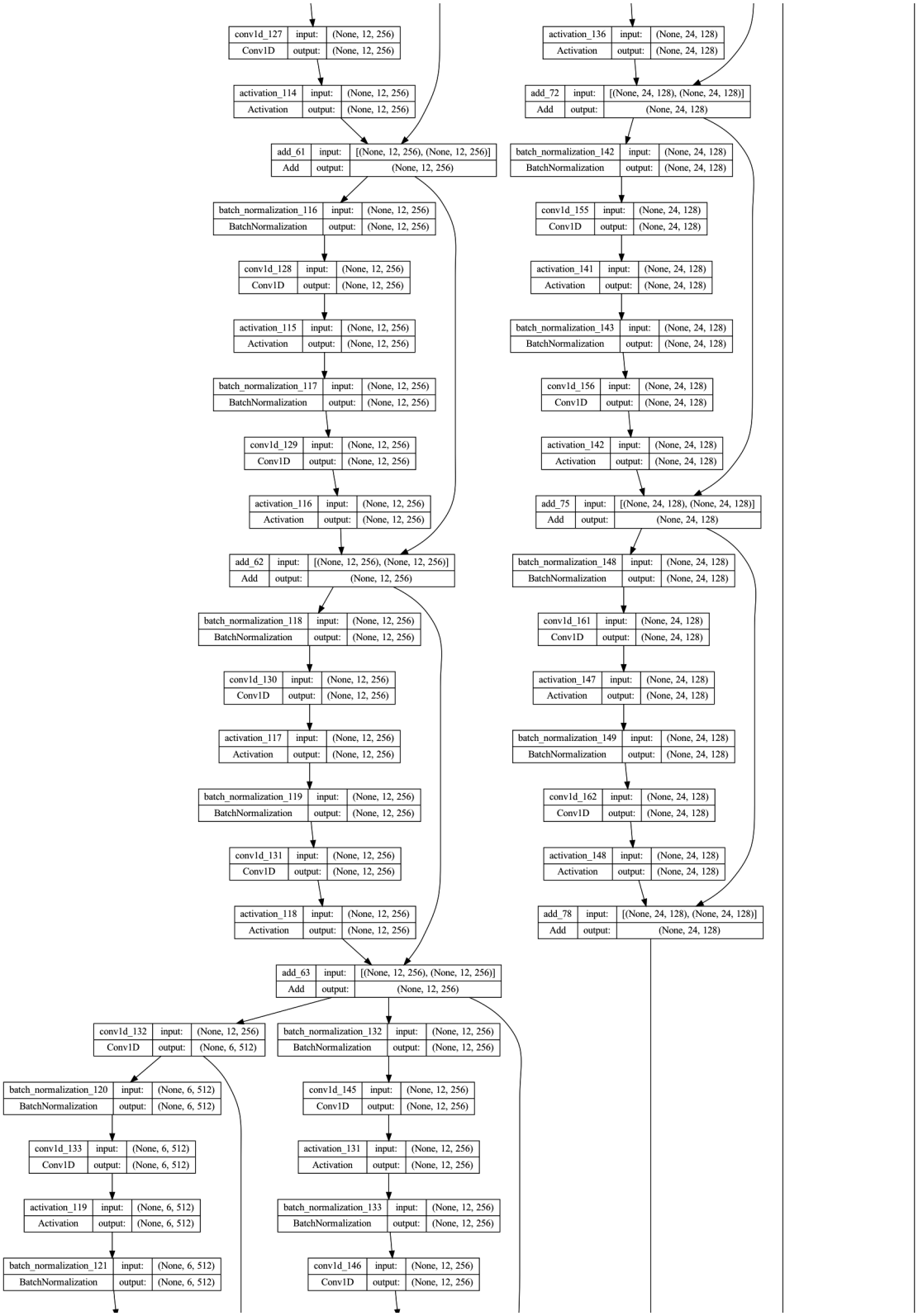


Figure 7.3: DGN for molecular finding; generator 2:

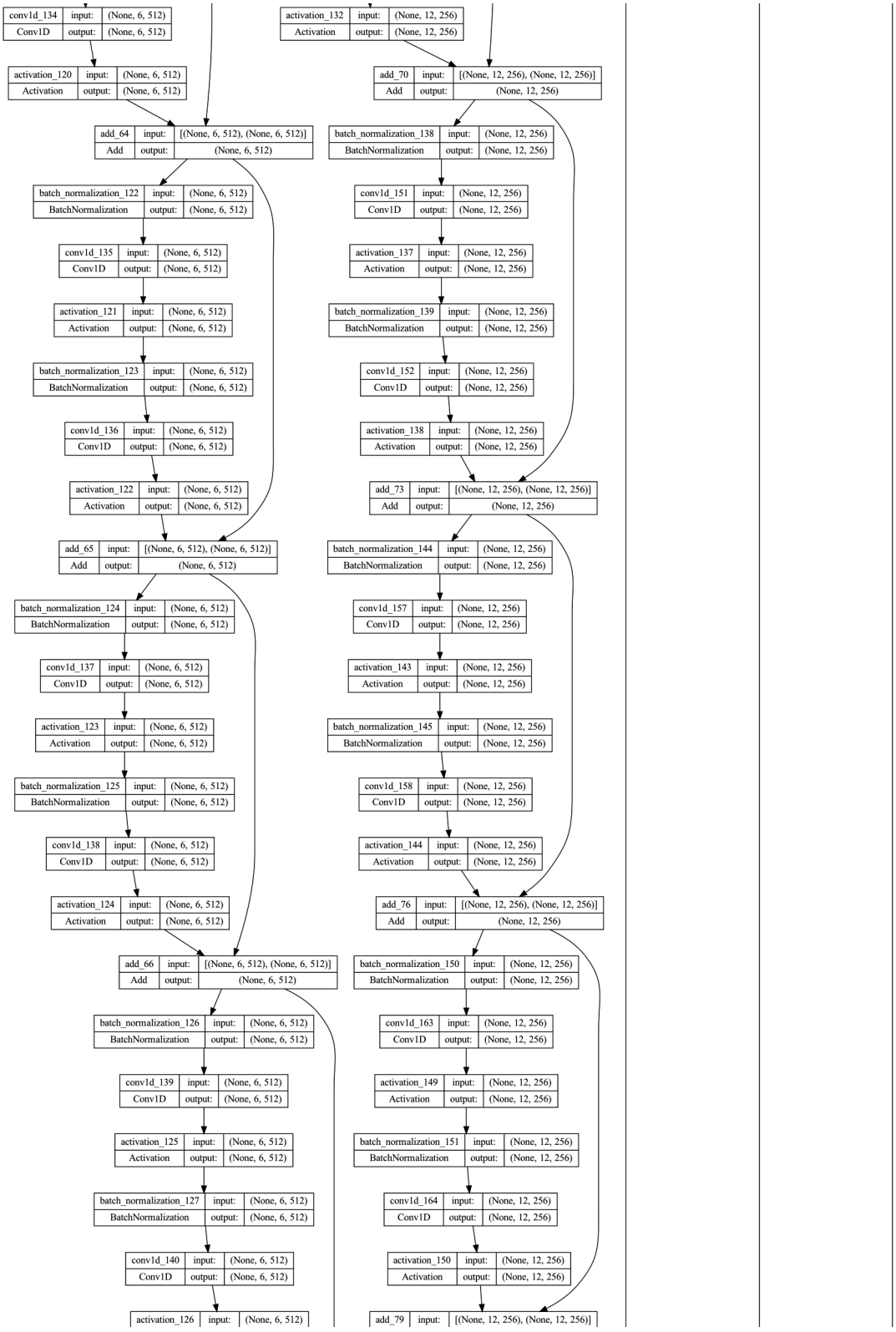


Figure 7.4: DGN for molecular finding; generator 3:

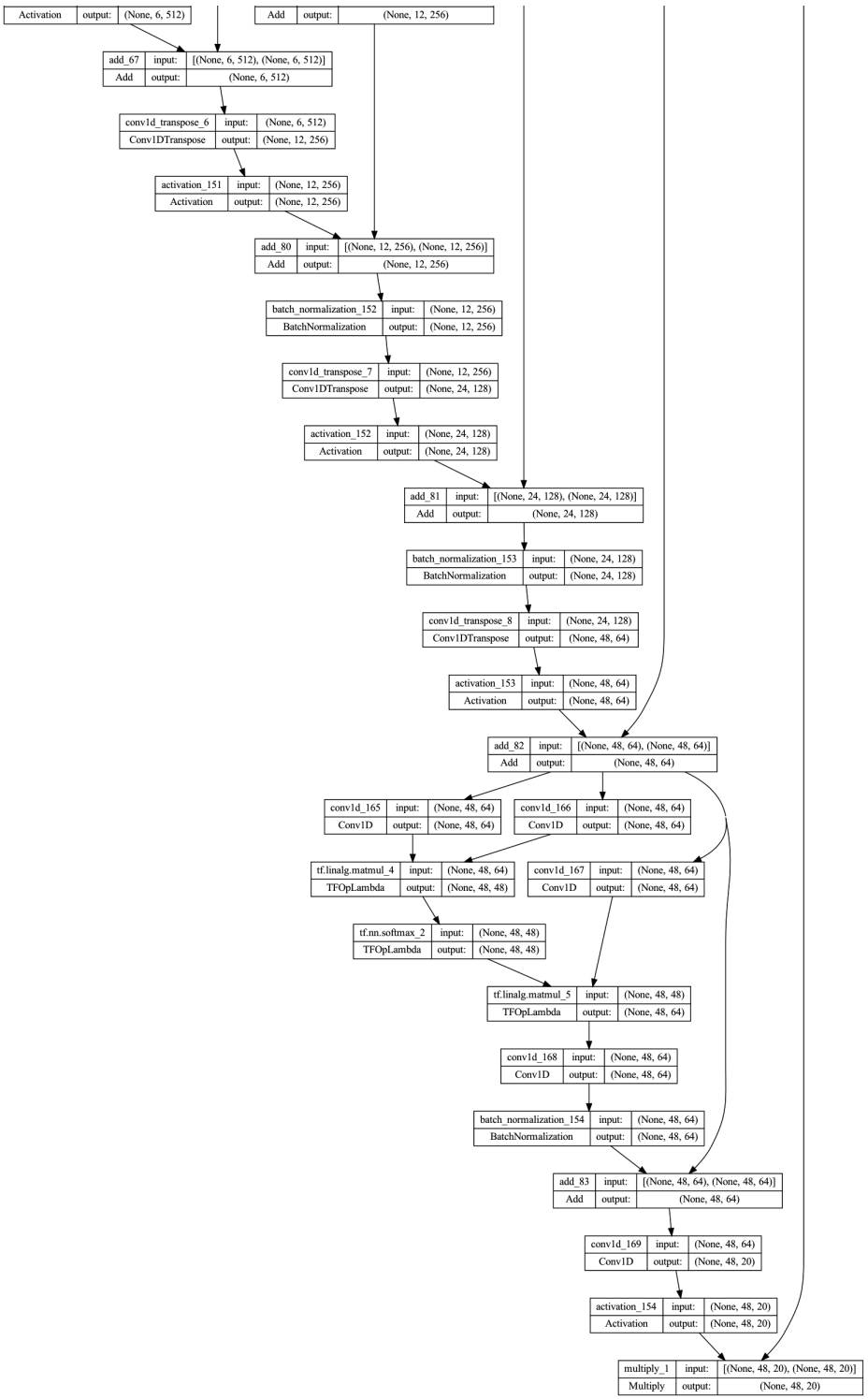


Figure 7.5: DGN for molecular finding; generator 4:

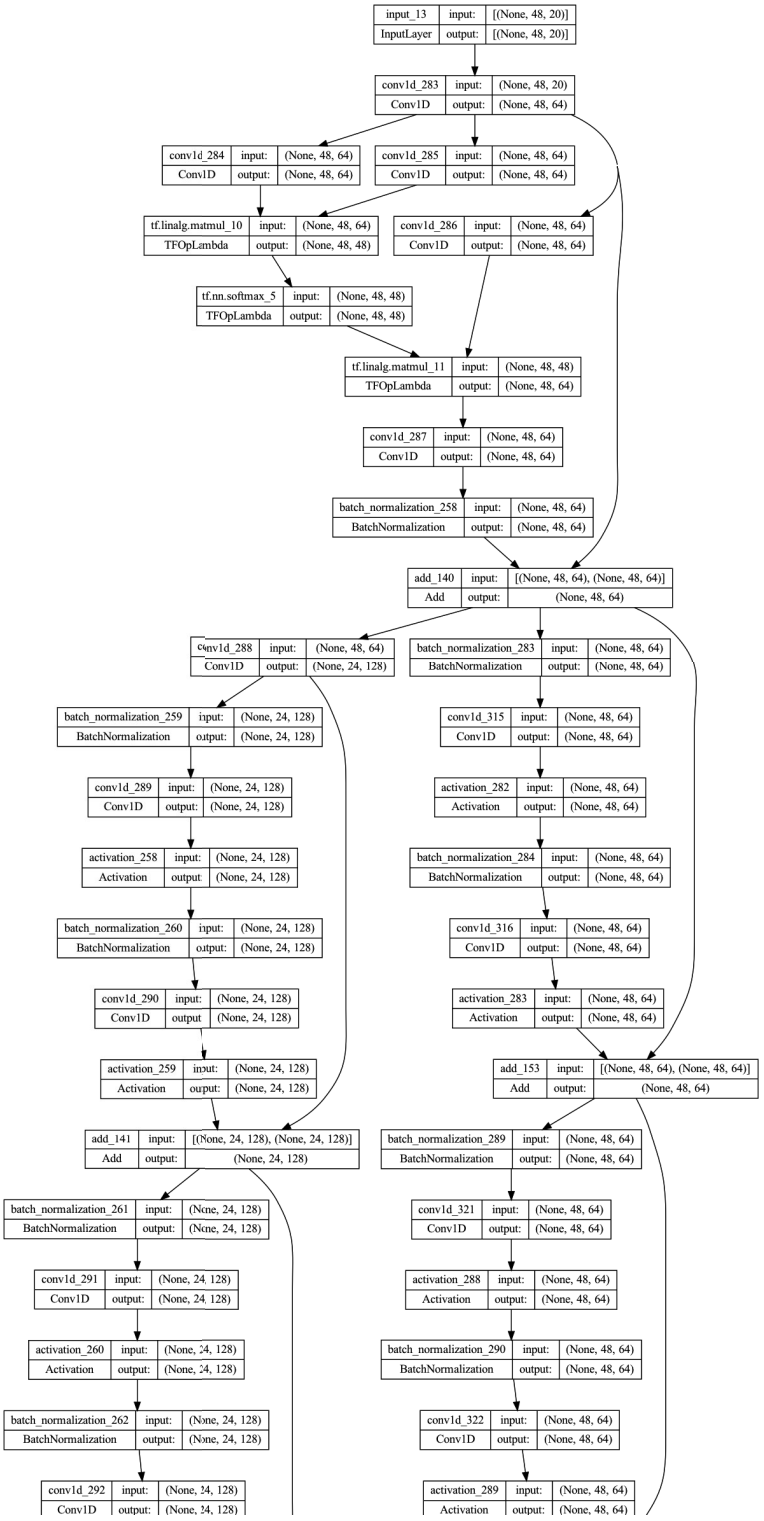


Figure 7.6: DGN for molecular finding; estimator 0:

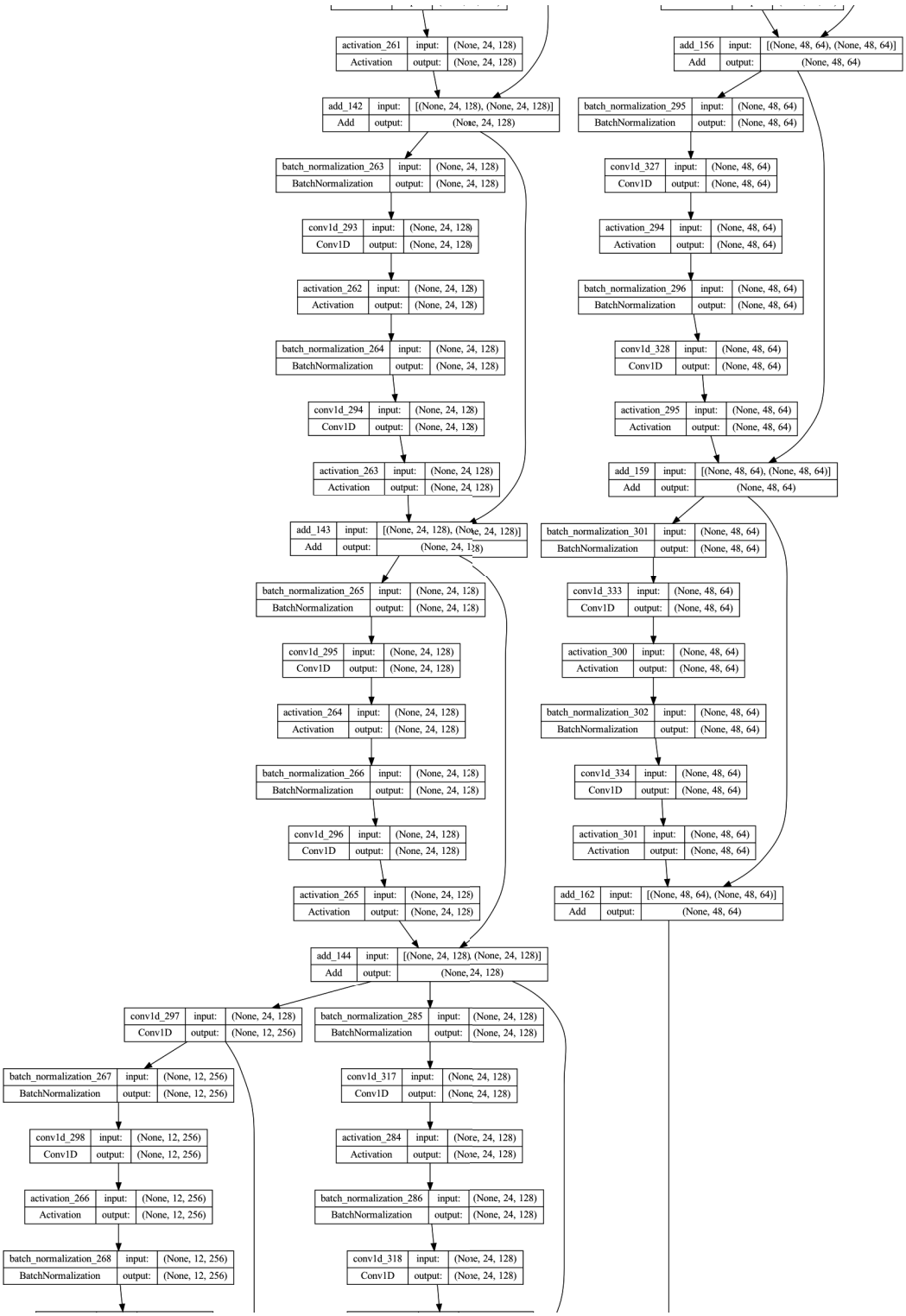


Figure 7.7: DGN for molecular finding; estimator 1:

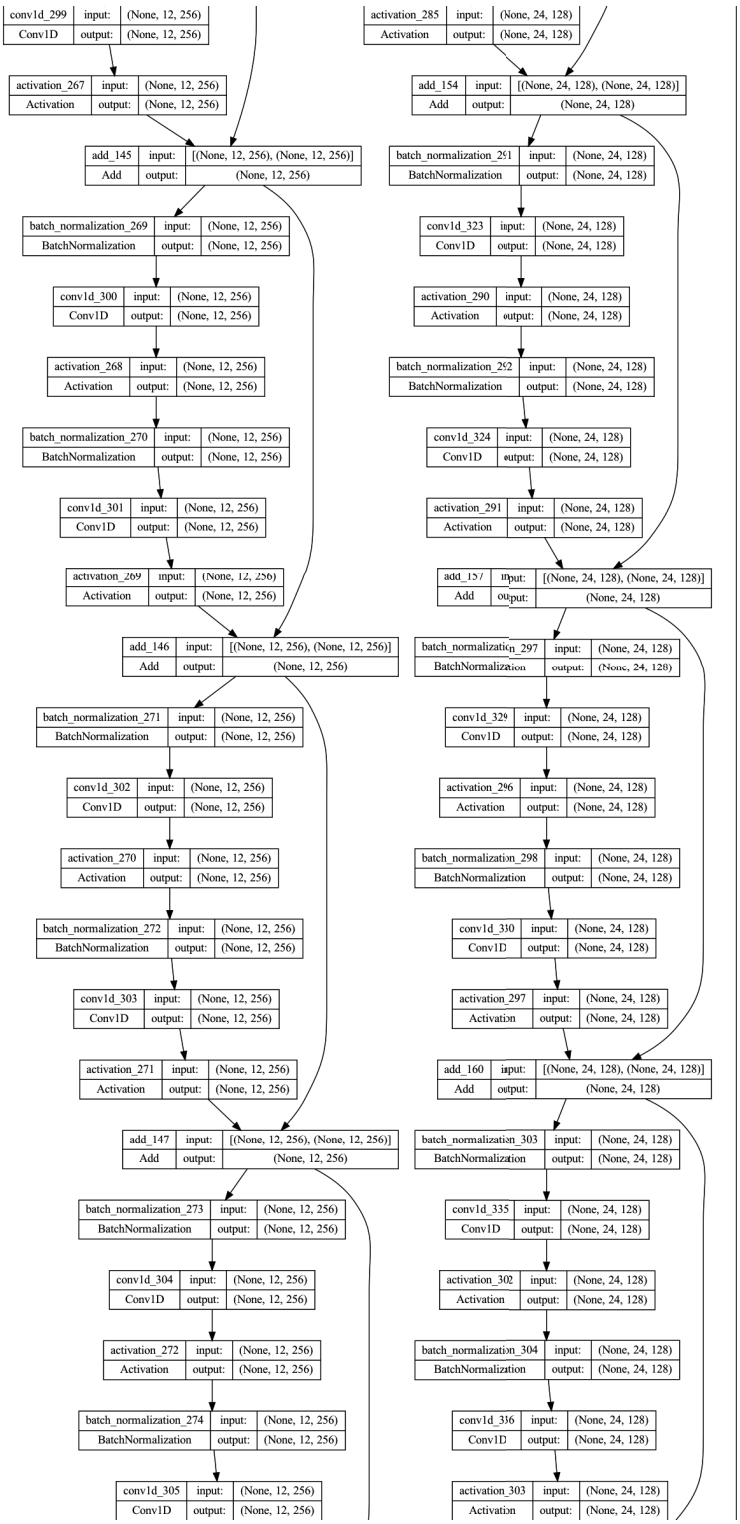


Figure 7.8: DGN for molecular finding; estimator 2:

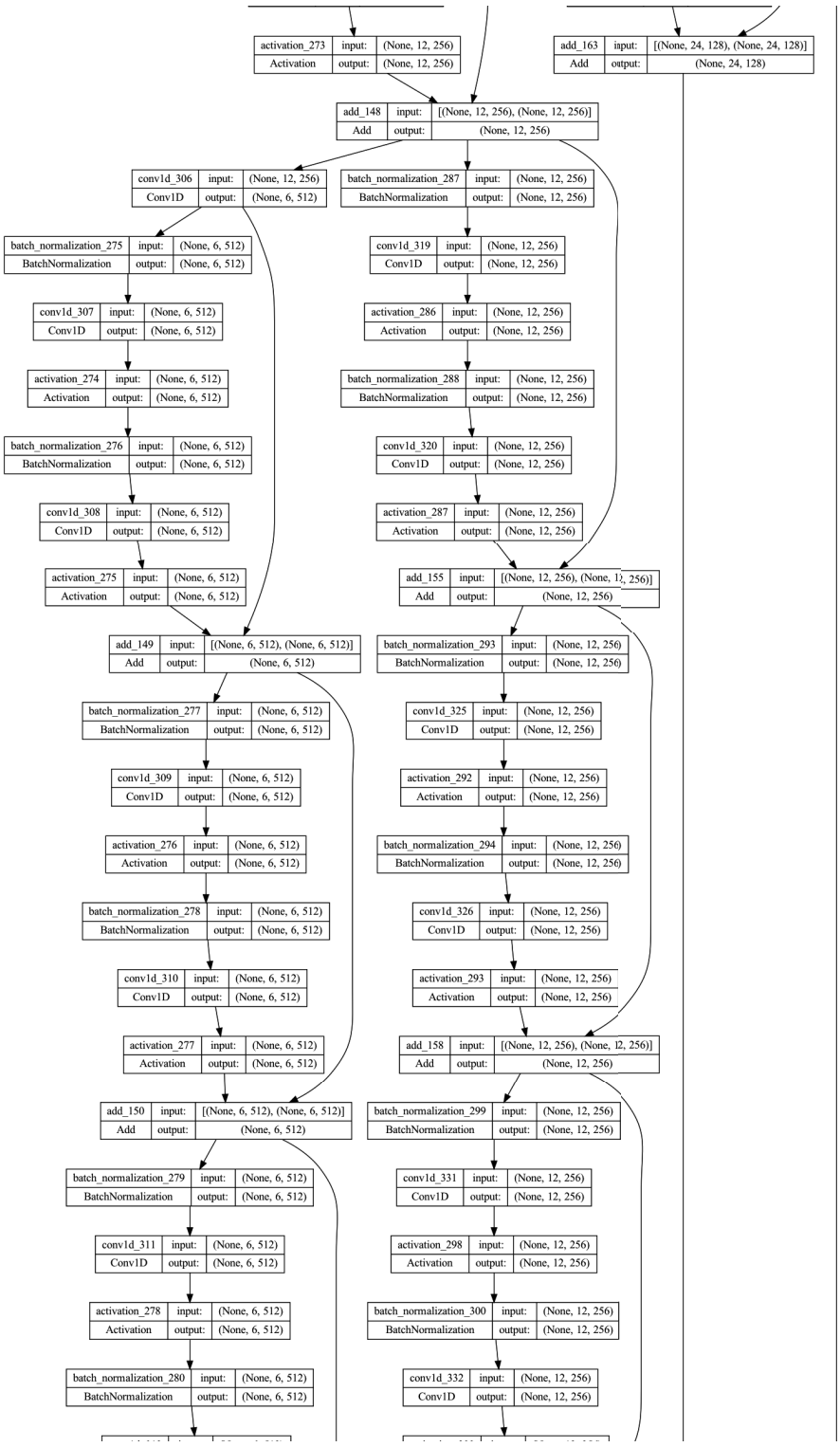


Figure 7.9: DGN for molecular finding; estimator 3:

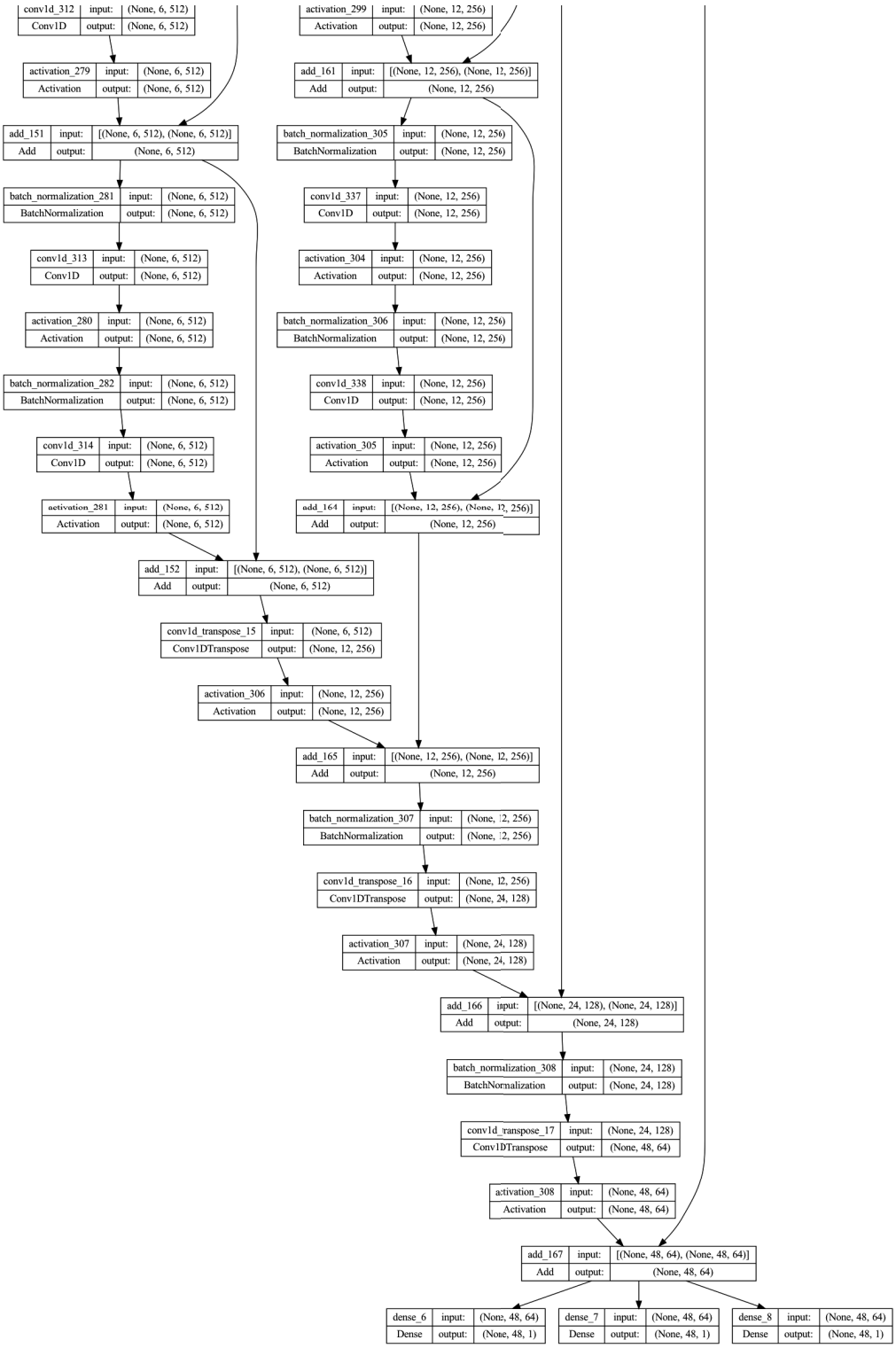


Figure 7.10: DGN for molecular finding; estimator 4:

Bibliography

[1] , . URL: https://keras.io/.

[2] , . URL: https://www.tensorflow.org.

[3] Arjovsky, M., Bottou, L., 2017. Towards principled methods for training generative adversarial networks. arXiv preprint arXiv:1701.04862 .

[4] Bäck, T., Fogel, D.B., Michalewicz, Z., 2018. Evolutionary computation 1: Basic algorithms and operators. CRC press.

[5] Beyer, H.G., Schwefel, H.P., 2002. Evolution strategies–a comprehensive introduction. Natural computing 1, 3–52.

[6] Bickerton, G.R., Paolini, G.V., Besnard, J., Muresan, S., Hopkins, A.L., 2012. Quantifying the chemical beauty of drugs. Nature chemistry 4, 90–98.

[7] Comer, J., Tam, K., 2001. Lipophilicity profiles: theory and measurement. Pharmacokinetic optimization in drug research: Biological, physicochemical, and computational strategies , 275–304.

[8] Cybenko, G., 1989. Approximation by superpositions of a sigmoidal function. Mathematics of control, signals and systems 2, 303–314.

[9] De Cao, N., Kipf, T., 2018. MolGAN: An implicit generative model for small molecular graphs. ICML 2018 workshop on Theoretical Foundations and Applications of Deep Generative Models URL: https://github.com/nicola-decao/MolGAN.

[10] Ertl, P., Schuffenhauer, A., 2009. Estimation of synthetic accessibility score of drug-like molecules based on molecular complexity and fragment contributions. Journal of cheminformatics 1, 1–11.

[11] Faizi, S.A., Yoshitomi, S., 2017. Application procedure for optimized placement of buckling restrained braces in reinforced concrete building structures. International Journal of Civil, Environmental, Structural, Construction and Architectural Engineering 11, 377–381.

[12] Falk, T., Mai, D., Bensch, R., Çiçek, Ö., Abdulkadir, A., Marrakchi, Y., Böhm, A., Deubner, J., Jäckel, Z., Seiwald, K., et al., 2019. U-net: deep learning for cell counting, detection, and morphometry. Nature methods 16, 67–70.

[13] Fei, Y., Liao, W., Zhang, S., Yin, P., Han, B., Zhao, P., Chen, X., Lu, X., 2022. Integrated schematic design method for shear wall structures: a practical application of generative adversarial networks. Buildings 12, 1295.

[14] Goodfellow, I., Bengio, Y., Courville, A., 2016. Deep Learning. Adaptive Computation and Machine Learning series, MIT Press. URL: http://www.deeplearningbook.org/.

[15] Goodfellow, I.J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y., 2014. Generative adversarial networks. arXiv:1406.2661.

[16] Guimaraes, G.L., Sanchez-Lengeling, B., Outeiral, C., Farias, P.L.C., Aspuru-Guzik, A., 2017. Objective-reinforced generative adversarial networks (organ) for sequence generation models. arXiv preprint arXiv:1705.10843 .

[17] He, K., Zhang, X., Ren, S., Sun, J., 2016. Deep residual learning for image recognition, in: Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770–778.

[18] Holland, J., 1992. Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. Complex Adaptive Systems, MIT Press.

[19] Hornik, K., 1991. Approximation capabilities of multilayer feedforward networks. Neural networks 4, 251–257.

[20] Ioffe, S., Szegedy, C., 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv:1502.03167.

[21] Kingma, D.P., Ba, J., 2014. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 .

[22] Kingma, D.P., Welling, M., 2022. Auto-encoding variational bayes. arXiv:1312.6114.

[23] Kipf, T.N., Welling, M., 2016. Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907 .

[24] The Organization of the Ministry of Land, Infrastructure, T., Tourism (Eds.), 2020. Commentary on Technical Standards for Building Structures. National Gazette Sales Cooperative.

[25] Landrum, G., 2006. Rdkit open source toolkit for cheminformatics. URL: https://www.rdkit.org.

[26] Lee, Y., Choi, G., Yoon, M., Kim, C., 2021. Genetic algorithm for constrained molecular inverse design. arXiv:2112.03518.

[27] Mnih, V., Heess, N., Graves, A., et al., 2014. Recurrent models of visual attention. Advances in neural information processing systems 27.

[28] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al., 2015. Human-level control through deep reinforcement learning. nature 518, 529–533.

[29] O'Boyle, N., Dalke, A., 2018. Deepsmiles: an adaptation of smiles for use in machine-learning of chemical structures .

[30] Pascanu, R., Mikolov, T., Bengio, Y., 2013. On the difficulty of training recurrent neural networks, in: International conference on machine learning, Pmlr. pp. 1310–1318.

[31] Polishchuk, P.G., Madzhidov, T.I., Varnek, A., 2013. Estimation of the size of drug-like chemical space based on gdb-17 data. J Comput Aided Mol Des 27, 675–679. doi:10.1007/s10822-013-9672-4.

[32] Rester, U., 2008. From virtuality to reality - virtual screening in lead discovery and lead optimization: a medicinal chemistry perspective. Curr Opin Drug Discov Devel 11, 559–568.

[33] Ronneberger, O., Fischer, P., Brox, T., 2015. U-net: Convolutional networks for biomedical image segmentation, in: Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18, Springer. pp. 234–241.

[34] Samanta, B., De, A., Jana, G., Gómez, V., Chattaraj, P.K., Ganguly, N., Gomez-Rodriguez, M., 2020. Nevae: A deep generative model for molecular graphs. The Journal of Machine Learning Research 21, 4556–4588.

[35] Sheridan, R.P., Kearsley, S.K., 2002. Why do we need so many chemical similarity search methods? Drug Discov Today 7, 903–911. doi:10.1016/s1359-6446(02)02411-x.

[36] Smith, P., 2016. Structural design of buildings. John Wiley & Sons.

[37] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R., 2014. Dropout: a simple way to prevent neural networks from overfitting. The journal of machine learning research 15, 1929–1958.

[38] Storn, R., Price, K., 1997. Differential evolution-a simple and efficient heuristic for global optimization over continuous spaces. Journal of global optimization 11, 341.

[39] Sutton, R.S., Barto, A.G., 2018. Reinforcement learning: An introduction. MIT press.

[40] Watkins, C.J., Dayan, P., 1992. Q-learning. Machine learning 8, 279–292.

[41] Watkins, C.J.C.H., 1989. Learning from delayed rewards .

[42] Weininger, D., 1988. Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. Journal of chemical information and computer sciences 28, 31–36.

[43] Wildman, S.A., Crippen, G.M., 1999. Prediction of physicochemical parameters by atomic contributions. Journal of chemical information and computer sciences 39, 868–873.

[44] Wolpert, D.H., Macready, W.G., 1997. No free lunch theorems for optimization. IEEE transactions on evolutionary computation 1, 67–82.

[45] Wongprasert, N., Symans, M., 2004. Application of a genetic algorithm for optimal damper distribution within the nonlinear seismic benchmark building. Journal of Engineering Mechanics 130, 401–406.

[46] Zhang, H., Goodfellow, I., Metaxas, D., Odena, A., 2019. Self-attention generative adversarial networks, in: International conference on machine learning, PMLR. pp. 7354–7363.

Publications

This study includes portions of the following publications;

- Ito, Y. & Nguyen, L. M.. Directional Generative Networks. submitted to Neurocomputing 2023.
- Ito, Y. & Nguyen, L. M.. Directional Generative Networks; comparison to Evolutionary Algorithms, using measurements for molecules. submitted to SCIDOCA 2023.
- Ito, Y.. (2018). DESIGN SUPPORT DEVICE, STRUCTURE PRODUCTION METHOD, AND PROGRAM.
  patent publication number JP2019-191954
  `https://www.j-platpat.inpit.go.jp/c1800/PU/JP-2019-191954/15CB33DB3E204234A4E7E402EFC39CA484880BB6E76AD0B08BE5F020E6A20D73/11/ja`
- Ito, Y. & Nguyen, L. M.. (2020). STRUCTURAL DESIGN SUPPORT PROGRAM, STRUCTURAL DESIGN SUPPORT APPARATUS, AND STRUCTURAL DESIGN SUPPORT METHOD.
  patent publication number JP2022-013325
  `https://www.j-platpat.inpit.go.jp/c1800/PU/JP-2022-013325/B8E06E449BA5B01BD220D67B9DB96DE729CBE8A1D6C3DF9AE902A27B127DFFD0/11/ja`