Impact of Duplicating Small Training Data on GANs

Yuki Eizuka[1], Kazuo Hara[1] and Ikumi Suzuki[2]
[1]*Yamagata University, 1-4-12 Kojirakawa-machi, Yamagata City, 990-8560, Japan*
[2]*Nagasaki University, 1-14 Bunkyo, Nagasaki City, 852-8521, Japan*

Keywords: Generative Adversarial Networks, Small Training Data, Emoticons.

Abstract: Emoticons such as (^_^) are face-shaped symbol sequences that are used to express emotions in text. However, the number of emoticons is miniscule. To increase the number of emoticons, we created emoticons using SeqGANs, which are generative adversarial networks for generating sequences. However, the small number of emoticons means that few emoticons can be used as training data for SeqGANs. This is concerning because as SeqGANs underfit small training data, generating emoticons using SeqGANs is difficult. To address this problem, we duplicate the training data. We observed that emoticons can be generated when the duplication magnification is of an appropriate value. However, as a trade-off, it was also observed that SeqGANs overfit the training data, i.e., they produce emoticons that are exactly the same as the training data.

1 INTRODUCTION

In recent years, multi-layer neural networks (MNNs) have contributed to the considerable development of artificial intelligence. In particular, remarkable progress has been made in the technology used for generating data such as images, texts, and music using MNNs. The most representative of these are generative adversarial networks (GANs) (Goodfellow et al., 2014). GANs build models for data at hand describing the data generation mechanism. Subsequently, new images, texts, music, among others, can be created using the model.

In this study, we use GANs to create emoticons (precisely, kaomoji). An emoticon is a sequence of symbols that make up the shape of a face. It is often used to express emotions such as laughter, sadness, and anger in blogs or social networking site (SNS) texts. However, currently, the number of emoticons is miniscule. For example, only 95 emoticons representing laughter are in the SHIMEJI dictionary,[1] as shown in Table 1. Nowadays, SNSs are widely used, and the demand for emotional expressions is on an increase. The objective of this paper is to increase the number of emoticons by automatically generating emoticons using GANs.

GANs are used to build a generator reproducing the characteristics of the data at hand, which is called the original data or training data. To achieve this, GANs employ a discriminator as a guide. GANs are algorithms that alternately update a generator and discriminator by repeating the following three steps. First, in step 1, data are generated using a current generator, which is called fake data (see Figure 1(a)). Next, in step 2, we build a discriminator that distinguishes the fake data from the original data (Figure 1(b)). A discriminator was used to evaluate the performance of the generator. If the fake data and the original data cannot be discriminated by the discriminator, the generator is successfully built; that is, the data with the characteristics of the original data are successfully produced. However, if the fake data and the original data can be discriminated, we proceed to step 3, where the generator is rebuilt. Ideally, by repeating the three steps, we obtain a generator that can produce data indistinguishable from or very similar to the original data (Figure 1(c)).

In GANs, the main role of the discriminator is to guide the rebuilding of the generator. That is, a new generator is built by being guided by the discriminator such that the generator produces fake data that are difficult for the discriminator to distinguish from the original data.

More specifically, the discriminator divides the data space into two regions:

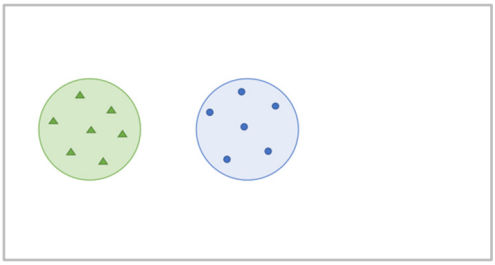
- positive region, where the original data is likely to be distributed, and
- negative region, where the original data is unlikely to be distributed.

[1] https://simeji.me/blog/顔文字-一覧/kaomoji

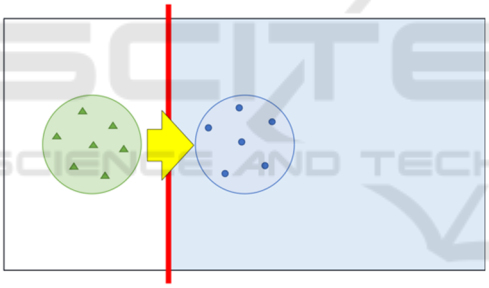
Eizuka, Y., Hara, K. and Suzuki, I.
Impact of Duplicating Small Training Data on GANs.
DOI: 10.5220/0010583403080315
In *Proceedings of the 10th International Conference on Data Science, Technology and Applications (DATA 2021)*, pages 308-315
ISBN: 978-989-758-521-0

Table 1: Examples of emoticons representing laughter.

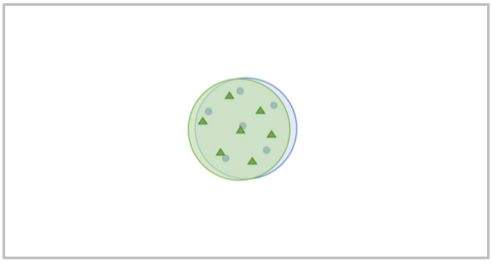
(∗´ ∀ `)	(ˆˆ)	(ˆ▽ˆ)o	(ˆ−ˆ)
(· ∀·)	(∗˚▽˚)ﾉ	(ˆ0ˆ)	(ˆДˆ)
(￣∀￣)	(´ ▽ `)	(⊙ㅂ⊙)	(≧∇≦)
(⌒▽⌒)	(° ▽°)	(☆∀☆)	(ŏ 艸 ŏ)
(((ˆ−ˆ)))	(★‿★)	(∗ˆ ω ˆ∗)	(∗ˆˆ)v



(a) The blue circle is the region where the original data are distributed and the points inside the blue circle represent the original data. The green circle is the region where the data that will be generated by a current generator are distributed and the points inside the green circle are fake data.



(b) The red line is the separation boundary determined by the discriminator, and it divides the data space into positive and negative regions. The positive region is indicated in blue. A new generator is rebuilt by moving toward the positive region.



(c) If the algorithm works, the green circle approaches the blue circle.

Figure 1: An overview of GANs algorithm.

Based on the region determined by the discriminator, a new generator is built in step 3 by moving toward the positive region (see Figure 1(b)).

However, the small number of existing emoticons means that the training data available for GANs to create emoticons is also small. If there are only a few training data, repeating the three steps may not build the desired generator. In fact, we herein reports that vanilla GANs (precisely, vanilla SeqGANs) do not fit small training data and thus fail to generate emoticons.

We believe that the reason GANs fail when there are only a few training data is that the positive region determined by the discriminator is too large to guide the generator. Therefore, even if the generator moves toward the positive region, the distribution of the data generated by the generator does not necessarily approach the distribution of the original data (see Figure 2).

To cope with this problem, we build a discriminator by improving step 2 such that the positive region determined by the discriminator includes only the region where the original data are distributed (see Figure 3). Specifically, we duplicate the original data N times and use them to build the discriminator. We expect that the larger the value of N is, the more the positive region will contain only the neighborhood of the original data. As a result, GANs will be successful in generating the desired data, i.e., new emoticons.

2 BACKGROUND

We herein use SeqGANs (Yu et al., 2017) to create emoticons. To explain SeqGANs, we first describe discriminators and generators as MNNs, which are the basic components of GANs. Next, we explain the GANs that build generators using discriminators as an auxiliary. Finally, we describe SeqGANs, which are GANs for generating symbol sequences.

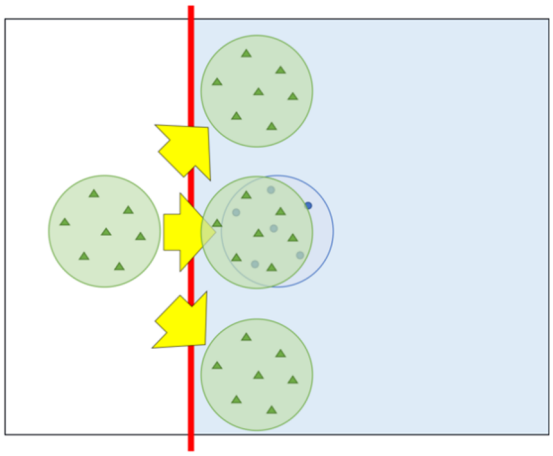


Figure 2: If the positive region is wide, the generator can move in various directions, including the direction in which the original data is not distributed.

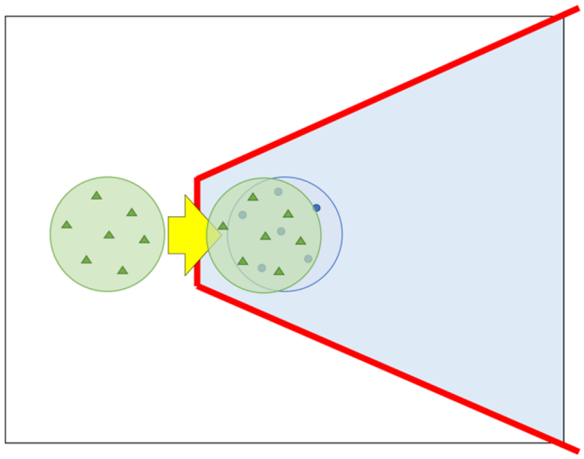


Figure 3: We assume that by narrowing the positive region, the generator can be moved to the correct direction.

2.1 Discriminators

When the purpose of MNNs is discrimination, the input layer receives data to be discriminated, such as text as a word sequence and emoticons as a symbol sequence.

For example, if we want to distinguish whether the content of the received email is "spam" or "non-spam," the email text is entered in the input layer. Subsequently, in the intermediate layers, feature extraction almost equivalent to the nonlinear principal component analysis is performed, and the input data are converted to a feature vector. Finally, in the output layer, a calculation similar to logistic regression was performed, and the feature vector was converted to a probability value ("non-spam" probability) between 0.0–1.0. If the output value is less than 0.5, it is judged to be "spam," and if it is 0.5 or more, it is judged to be "non-spam."

In summary, a discriminator constructed by an MNN is a function $y = D(x, \theta_d)$ that returns the probability value y for the input data x with θ_d as the parameter of the MNN. Tuning the parameter θ_d of the discriminator with a set of pairs of input data x and its correct output y is called learning the discriminator.

2.2 Generators

When the purpose of an MNN is to generate, the input layer receives a random vector. The input random vector is nonlinearly transformed as it passes through the intermediate layers; finally, at the output layer, data such as text and emoticons are outputted. In other words, a generator constructed by an MNN is a function $x = G(z, \theta_g)$ that returns data x for a random input vector z with θ_g as the parameter of the MNN.

When generating a sequence of symbols, the input layer receives a one-hot vector corresponding to a symbol that represents the Begin of Sentence. Then, symbols are randomly generated one by one to form a symbol sequence according to a multinomial distribution wherein the number of terms in the multinomial distribution is the number of symbols. The multinomial distribution is constructed using an MNN with parameter θ_g.

2.3 Generative Adversarial Networks

How can we build a generator $G(z, \theta_g)$ that produces data with the desired characteristics rather than random data? GANs are the answer to this question:

To produce the desired data using GANs, we first prepare a set of data x_{real} having the desired characteristics, which are called original data or training data for GANs. Next, by providing an initial value to parameters θ_d and θ_g, we repeat steps 1 to 3 as follows:

Step 1. Generate a set of fake data x_{fake} using generator $G(z, \theta_g)$ with a random vector z.

Step 2. Update the parameter θ_d of discriminator $D(x, \theta_d)$ such that the set of generated fake data x_{fake} is discriminated from the set of original data x_{real}.

Step 3. Update the parameter θ_g of generator $G(z, \theta_g)$ to generate fake data x_{fake} in the positive region determined by the discriminator.

More formally, in Step 2, the discriminator parameter θ_d is updated such that the log-likelihood of θ_d given data x_{real} and x_{fake} increases. The log-likelihood was calculated as follows:

$$\begin{aligned}&\log likelihood(\theta_d|x_{real}, x_{fake})\\&= \log D(x_{real}, \theta_d) + \log\left(1 - D(x_{fake}, \theta_d)\right).\end{aligned} \quad (1)$$

In Step 3, the generator parameter θ_g is updated such that the log-likelihood of θ_g given data $x_{fake} = G(z, \theta_g)$ decreases. The log-likelihood was calculated as follows:

$$\begin{aligned}&\log likelihood\left(\theta_g|x_{fake}\right)\\&= \log\left(1 - D(x_{fake}, \theta_d)\right)\\&= \log\left(1 - D\left(G(z, \theta_g), \theta_d\right)\right).\end{aligned} \quad (2)$$

Here, the discriminator $D(x, \theta_d)$ is a function that outputs the probability that data x is the original data. Conversely, $1 - D(x, \theta_d)$ denotes the probability that the discriminator predicts the data x as fake data. This means that, in Step 2, the discriminator parameter θ_d is updated such that the discriminator correctly discriminates between the original data and the fake data with a high probability. In Step 3, the generator parameter θ_g is updated such that the fake data generated by the generator are judged as fake data by the discriminator with a low probability.

If all goes well, the parameter θ_g is tuned such that $P_g(x|\theta_g)$ approaches $P_{true}(x|\theta_{true})$, where $P_{true}(x|\theta_{true})$ denotes the distribution of data x sampled from the original data, and $P_g\left(x|\theta_g\right)$ denotes the distribution of data x generated by the function $G(z, \theta_g)$.

Mathematically, GANs are equivalent to solving the minimax problem of an objective function $f(\theta_d, \theta_g)$ for the discriminator parameter θ_d and generator parameter θ_g. The objective function $f(\theta_d, \theta_g)$ is obtained by marginalizing the $\log likelihood(\theta_d|x_{real}, x_{fake})$ with the distribution $P_{true}(x_{real}|\theta_{true})$ of the original data and the distribution $P_g\left(x_{fake}|\theta_g\right)$ of the generated data. Thus, the objective function of the GANs is

$$\begin{aligned}&f(\theta_d, \theta_g)\\&= E_{P_{true}}[E_{P_g}[\log likelihood(\theta_d|x_{real}, x_{fake})]]\\&= E_{P_{true}}[E_{P_g}[\log D(x_{real}, \theta_d)\\&\quad + \log\left(1 - D(x_{fake}, \theta_d)\right)]]\\&= E_{P_{true}}[\log D(x_{real}, \theta_d)]\\&\quad + E_{P_g}[\log\left(1 - D\left(G(z, \theta_g), \theta_d\right)\right)].\end{aligned} \quad (3)$$

The GANs algorithm calculates the following equation:

$$\arg\min_{\theta_g}\max_{\theta_d} f(\theta_d, \theta_g). \quad (4)$$

In other words, in Step 2, θ_d is updated such that $f(\theta_d, \theta_g)$ becomes large while fixing θ_g. In Step 3, θ_g is updated such that $f(\theta_d, \theta_g)$ becomes small while fixing θ_d.

Figure 4 shows an image of the objective function of GANs, showing the values of $f(\theta_d, \theta_g)$ as a contour map where the horizontal axis is θ_d and the vertical axis is θ_g. The value of $f(\theta_d, \theta_g)$ is large in the upper left and lower right and small in the lower left and upper right of the figure. The center of the figure is a saddle point, which is the solution to the minimax problem.

In Figure 4, each small rectangle represents the space of data x. The red line is the separation boundary determined by the discriminator $D(x, \theta_d)$, and it divides the space of data x into positive and negative regions. The positive region is indicated in blue. The blue circle is the region supported by $P_{true}(x|\theta_{true})$, and the points inside the blue circle represent the original data x_{real}. The green circle is the region supported by the distribution $P_g\left(x|\theta_g\right)$ of data x generated by the generator function $G(z, \theta_g)$, and the points inside the green circle are fake data x_{fake}. The black arrow represents the transition of θ_d and θ_g by the iterative algorithm of the GANs. The green circle of $P_g\left(x|\theta_g\right)$ approaches the blue circle of $P_{true}(x|\theta_{true})$ by repeatedly updating θ_d and θ_g from the initial value in the upper left and converging to the center, i.e., the solution of the minimax problem.

2.4 SeqGANs

SeqGANs generate sequence data in a GAN framework. SeqGANs differ from GANs in that they solve a specific problem occurring in the update of θ_g in Step 3. The problem is that the generator parameter θ_g can be significantly updated in response to slight differences in the sequence data.

For example, in generating emoticons, $D(x_{fake}, \theta_d) \approx 1$ for $x_{fake} =$ (^____^), which is almost the original data. However, $D(x_{fake}, \theta_d) \approx 0$ for $x_{fake} =$ (^____^_, even though there is only a slight difference from (^____^). This is because $x_{fake} =$ (^____^_ does not have a pair of left and right parentheses; hence, it is not recognized as an emoticon. In Step 3, for x_{fake}, where the value of

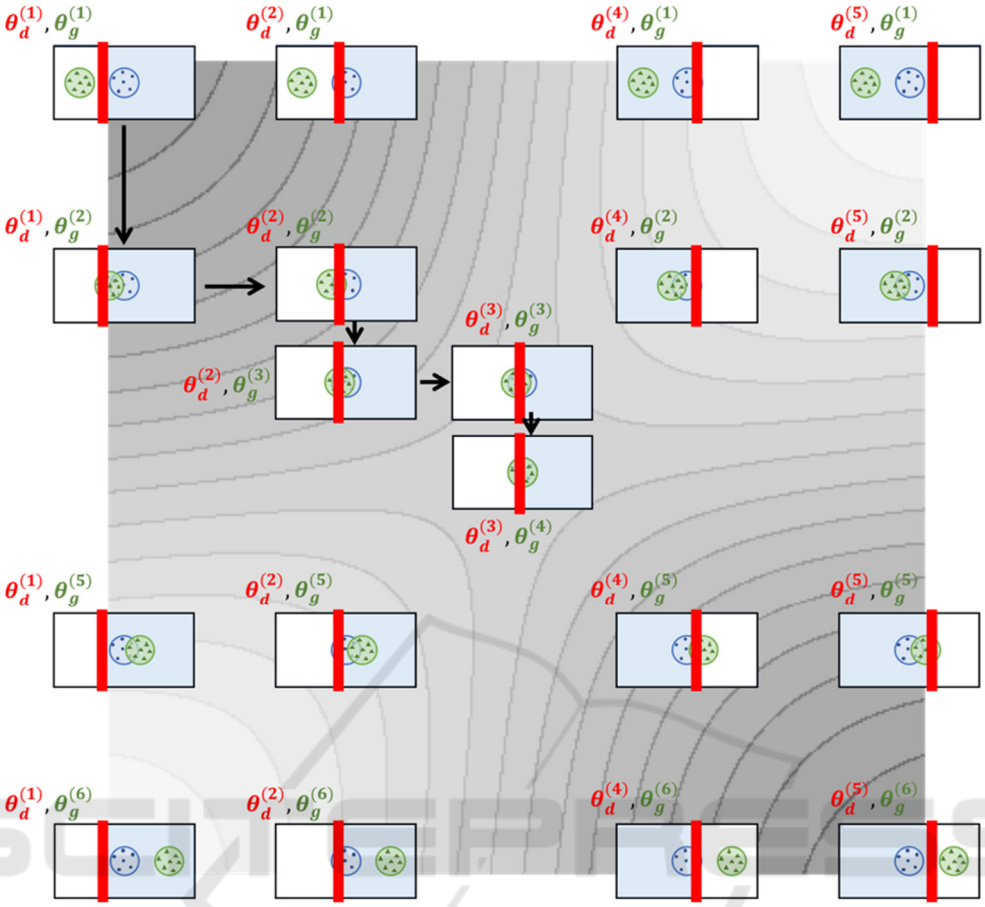


Figure 4: An image of the objective function of GANs, shown as a contour map. The black arrow represents the transition of θ_d and θ_g by the iterative algorithm of the GANs.

$D(x_{fake}, \theta_d)$ is close to 0, the generator parameter θ_g is updated such that the generator never generates x_{fake} again. Therefore, $x_{fake} = (\hat{}\_\_\_\_\hat{}\_$ will not be generated in the iterative algorithm in the future. This is not a good idea because it will become similar to the original data with a few more modifications.

To cope with this problem, for $x_{fake} = (\hat{}\_\_\_\_\hat{})$, SeqGANs evaluate partial sequences such as (, (^, ... , (^____^ by performing Monte Carlo simulations. Specifically, for each of partial sequences, SeqGANs randomly generate subsequent sequence until a complete sequence $x_{complete}$ is obtained. Subsequently, in Step 3, $D(x_{complete}, \theta_d)$ is calculated, and the parameter θ_g is updated using this. Thus, this problem is avoided.

3 DUPLICATE TRAINING DATA

In this study, SeqGANs were used to generate emoticons with only a small amount of original data. However, as seen in Figure 3, the discriminator cannot guide the generator in the correct direction because the positive region determined by the discriminator is too wide. To address this problem, we modify Step 2 in the GAN algorithm.

Specifically, the update formula of parameter θ_d with respect to the original data x_{real},

$$\theta_d \leftarrow \theta_d + \eta \frac{d}{d\theta_d} \log D(x_{real}, \theta_d) \quad (5)$$

is modified to

$$\theta_d \leftarrow \theta_d + N\,\eta \frac{d}{d\theta_d} \log D(x_{real}, \theta_d) \quad (6)$$

where η is the learning rate, and N is an integer greater than or equal to 1.

Table 2: Examples of symbol sequences generated by SeqGANs without duplicating the original data.

。ఓლ	ಡ ̂。	و✧ヾ)	๑و.)Д
︶o。 ッ▽)。 ̂)	̂	Ʊ(♫✌ ゚☆	ˊ≷★◠
∀Д٩	▽(̂ノ	ಡ▽✿ ̂	ξ ̀ ゚
ク◠。	ξ*◠✌ ª ノ	*ლ ̄ ̄))	̀
゚Ʊ∀)	!Ʊノ ̂))	★ఓ ゚)و๑。 *))	◡ ̂∀∀ ̀

Table 3: Examples of symbol sequences generated by SeqGANs when duplicating the original data N = 10 times. Those in red meet the definition of emoticons.

(́ ▽ ̀))	ヽヽ(*^o^*)／	♫ヽ(゚ ▽゚)ノ♩	(⌒艸⌒)
♫ヽ(゚ ▽゚ ♪	ヾ(*^▽^)/★*☆	(ლ▽^=)	(((o(*゚▽゚*)o))
ヾ(๑ ỏƱỏ ๑)″	(✿ ᴼ 艸 ᴼ ,,,,)	✧*٩(ˊωˋ*)و✧*。	ヽ(*^^)o
(ಡ∀・)	(*^▽^)/◎ク	л _ ɦ a ɦ a	(ｷ。✪ω✪ヽ(๑)✌✌
(o^^oo)	(✿◠>_Ʊ)	٩(*^^*)ノ	(๑ო ́ •)

By doing this, the value of $D(x_{real}, \theta_d)$ can be increased. Subsequently, we expect that the positive region determined by the discriminator will include only the support region of the distribution of the original data.

Simply put, if N is an integer greater than or equal to 1, this is equivalent to duplicating the original data N times and using the conventional update formula. Therefore, in the experiment presented herein, we duplicate the original data N times without changing the update formula. Furthermore, along with duplicating the original data N times, we increased the number of fake data generated by the generator N times and used them to train the discriminator.

4 EXPERIMENT

The purpose of the experiment was to generate emoticons representing laughter using SeqGANs. We used 95 emoticons in Table 1 as original data, i.e., the training data for SeqGANs. Both the generator and the discriminator in SeqGANs have a 128-dimensional embedding layer and an LSTM layer. The output layer is a softmax layer for the generator and sigmoid for the discriminator.

First, Table 2 shows the symbol sequences generated by SeqGANs without duplicating the original data. Almost all generated symbol sequences did not resemble emoticons. To evaluate this quantitatively, we define emoticons as follows:

A symbol sequence in which all of the following I, II, and III hold is defined as an emoticon.

I. There is a pair of parentheses on the left and right that correspond to the outline of the face.

II. Symbols correspond to the eyes inside the outline of the face. Specifically, it is the case in which either of the following two holds:

(a) Symmetrical symbols corresponding to the eyes.

(b) Symbols corresponding to the eyes exist asymmetrically, but an asymmetric pattern exists in the original data.

III. If a symbol corresponds to the nose or mouth, it exists between the symbols corresponding to the eyes.

Applying this definition to the symbol sequences in Table 2, we can see that there is no symbol sequence that satisfies the definition of emoticons.

Next, Table 3 shows the symbol sequences generated by SeqGANs when the original data were duplicated N = 10 times and used as training data.

Approximately 70% of the generated symbol sequences meet the definition of emoticons.

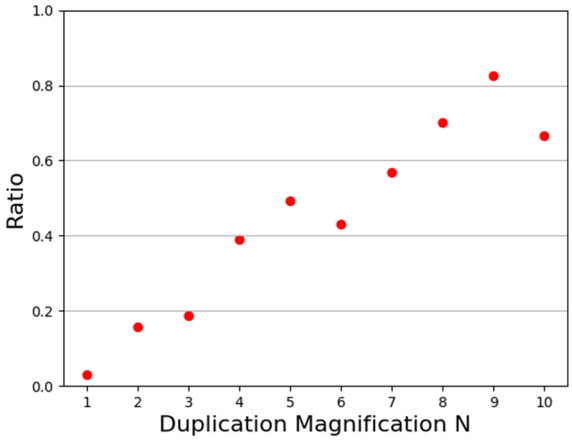


Figure 5: Ratio of the symbol sequences generated by SeqGANs that meet the definition of emoticons.

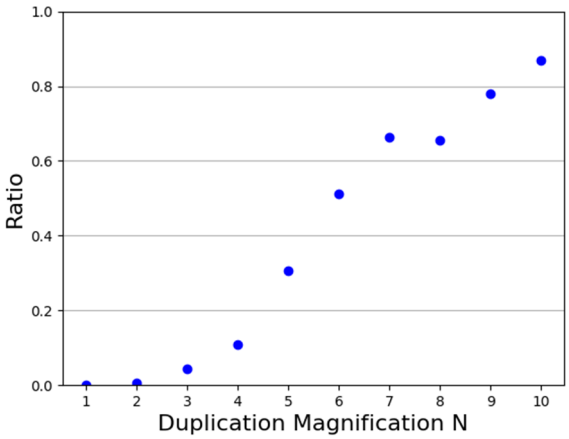


Figure 6: Ratio of the symbol sequences generated by SeqGANs that are exactly the same as the original data.

Figure 5 shows the change in the ratio of the symbol sequences generated by SeqGANs that meet the definition of emoticons when the duplication magnification N is moved from 1 to 10. As N increases, the ratio of symbol sequences that satisfy the definition of emoticons increases.

On the contrary, as the duplication magnification N increases, SeqGANs output emoticons that are exactly the same as the original data. In this experiment, if SeqGANs generated the same emoticons as the original data, the generation was redone. Figure 6 shows the ratio of the number of regenerated data to the number of generated data. It can be observed that the ratio increases as N increases. This is because when the duplication magnification N is high, the distribution of the data generated by the generator approaches the original data itself rather than the distribution of the original data. In other words, overfitting occurs.

5 RELATED WORK

Karras et al., (2020) focused on overfitting as a problem of GANs when training data was scarce. In contrast, we herein discussed a way to deal with underfitting to small training data.

Although GANs are excellent for building generators, GANs have a weakness in that tuning parameters is difficult. This is because, as shown in Figure 4, the solution of GANs is the saddle point of the objective function $f(\theta_d, \theta_g)$. The GAN algorithm usually repeats updating the parameters θ_d and θ_g alternately depending on the gradient of $f(\theta_d, \theta_g)$.

This calculation tends to be unstable because the objective function repeatedly increases and decreases. To stabilize the algorithm, we need to be aware of the following when updating the parameters:

(1) Is the length of the gradient vector appropriate, not too short or too long?

(2) Is the direction of the gradient vector pointing to the correct direction?

Regarding the issue (1), (Gulrajani et al., 2017) and (Mescheder et al., 2017) use the length of the gradient vector as a regularizer of the objective function. In contrast, we herein deal with issue (2).

6 CONCLUSION

We duplicated a small amount of training data and used it for SeqGANs. As a result, we successfully generated emoticons when duplication magnification was appropriately set. However, as a trade-off, it was also observed that SeqGANs overfit the training data, i.e., they produce emoticons that are exactly the same as the training data. Resolving this trade-off is a future task.

In this study, data generation was limited to emoticons representing laughter. However, there are also emoticons that express various meanings such as sadness, shy, anger, and surprise. Future tasks include creating emoticons that can distinguish between these subtle differences in meaning, and creating emoticons that have new meanings, such as shy + laughter.

REFERENCES

L. Mescheder, S. Nowozin, and A. Geiger, “The numerics of GANs,” In Proc. of the Advances in Neural Information Processing Systems, pp. 1825-1835, 2017.

I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative Adversarial Nets,” In Proc. of the Advances in Neural Information Processing Systems, pp. 2672—2680, 2014.

I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville, “Improved Training of Wasserstein GANs,” In Proc. of the Advances in Neural Information Processing Systems, pp. 5769–5779, 2017.

T. Karras, M. Aittala, J. Hellsten, S. Laine, J. Lehtinen, and T. Aila, “Training Generative Adversarial Networks with Limited Data,” In Proc. of the NeurIPS, 2020.

L. Yu, W. Zhang, J. Wang, and Y. Yu, “SeqGAN: Sequence Generative Adversarial Nets with Policy Gradient,” In Proc. of the AAAI Conference on Artificial Intelligence, vol. 31, no. 1, 2017.