The 33rd Annual Conference of the Japanese Society for Artificial Intelligence, 2019

Conditional DCGAN's Challenge: Generating Handwritten Character Digit, Alphabet and Katakana

Rina Komatsu[*1] Tad Gonsalves[*1]

[*1] Sophia University

Developing deep learning models has a great potential in assisting human tasks involving design and creativity. This study deals with generating handwritten characters using deep learning techniques. The task is not simply generating images randomly, but generating them conditionally, making a distinction according to the UI designates. To solve this task, we constructed the Conditional DCGAN model which includes the techniques from DCGAN and Conditional GAN. We tried training the models to be able to generate conditional images by adding label information as input to the Generator. Deep learning experiments were performed using 141319 training data consisting of 96 kinds of characters including digits, Roman alphabets and Katakana. The Generator trained by inputting random noise concatenated with the 96 kinds of characters, could generate each kind of character by just adding the appropriate label information.

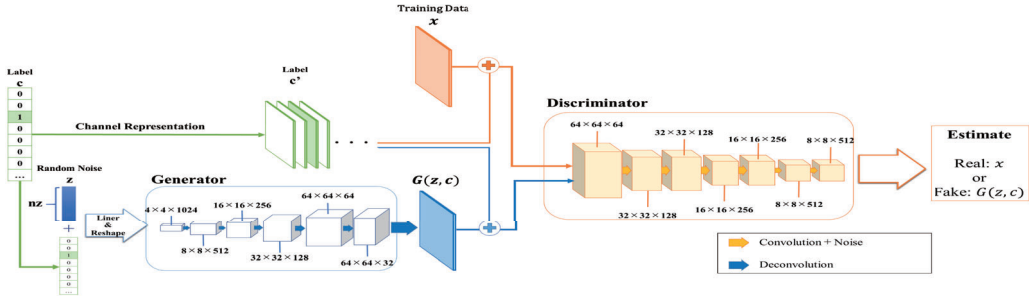

Figure 1. Proposed Conditional DCGAN which generates conditional handwritten characters

1. Introduction

Of late, more and more Deep Learning techniques which deal with generating images are been developed and as a result realistic images are being generated. In addition to being able to generate images that are realistic, the potential of deep learning is immense, such as supplementing blank areas and learning to imitate styles of famous painters to create artistic images.

To further test the potential of deep learning, we tried generating handwritten characters by developing a model called *Conditional DCGAN* which combines DCGAN with cGAN (Conditional GAN). A conditional label is added as additional input along with the image input to the model. The target kinds of character we dealt with in this study are not only digits, but also alphabets and katakana (Japanese script) and some special characters. The goal of this study is generating handwritten characters by making a distinction among more than 90 different kinds of them.

In our experiment, we obtained results by changing the dimensions of the random noise which is part of the input for the models. It can be inferred from our results that when the number of dimensions of noise falls below the number of labels, the model cannot generate images that are likely to be characters; and on the other hand, if the number of types exceeds 90, the model can generate the specified characters.

Contact: Rina Komatsu, Faculty of Science & Technology, Sophia University, Tokyo, Japan, r_komatsu@outlook.com

2. Related Work

We construct the model (the architecture shows in Figure 1) based on the techniques from DCGAN and cGAN. This section introduces generating method: GAN and DCGAN, also introduce cGAN to generate conditional images.

2.1 GAN & DCGAN

GAN: Generative Adversarial Net [Ian J. Goodfellow, 2014] is a generative network model that generates images by training a Generator and a Discriminator that are tied together in an adversarial relationship. The Generator plays the role of generating images from a given probability density distribution with random noise input, while the Discriminator plays the role of distinguishing the real input data from the fake data generated by the Generator. However, GAN has the weak points that the probability density distribution Generator learns is unable to indicate clearly and training Generator and Discriminator tend to unstable [Naoki Shimada et al, 2017].

DCGAN: Deep Convolutional Generative Adversarial Network [Alec Radford et al, 2015] is a generative model designed to solve this weak point by employing stable learning techniques such constructing fractional-strided convolution in Generator and strided convolution in Discriminator, in addition, instead of pooling layers, adapting batch normalization [Sergey

Ioffe et al, 2015] to each layer and so on. The Generator in DCGAN extends the information through upsampling from random noise input, while the Discriminator extracts feature maps through convolutions. As a result, DCGAN succeeds in generating more realistic images than GAN.

About how to calculate Loss GAN and DCGAN utilize Discriminator's output in loss function shown in formulae (1) and (2) to update each the parameters of each model. Formula (1) is loss function for Discriminator and (2) is the one for Generator. If the Discriminator learns good work in distinguishing, then log(D(x)) increases and 1-log(G(z)) decreases on the contrary. On the other hand, if the Generator reaches a matured level that deceives the Discriminator, then log(D(z)) increases.

$$L_D(G,D) = E\big[\log\big(D(x)\big)\big] + E[1 - \log(D(G(z)))] \quad (1)$$

$$L_G(G,D) = E[-\log(D(G(z)))] \quad (2)$$

where,
x is the training sample data, and
G(z) is the Generator's output from random noise z.

2.2 cGAN: Conditional GAN

cGAN: Conditional GAN [Mehdi Mirza et al, 2014] is the generative model which can output designated images by adding auxiliary information (represented as one-hot vector) such as a label corresponding with the kinds or modality to Generator after finishing training in the Generator and Discriminator.

Figure 2 shows a simplified structure of Conditional Adversarial Nets dealing with the auxiliary information. Random noise z and an auxiliary information y are input to the Generator combined forward to hidden layer. These jointed data help the Generator to suggest the probability density distribution to which the training sample data belongs. Also, training sample data *x* or generated ones G(*z*|*y*) and *y* are input to Discriminator combined in same.

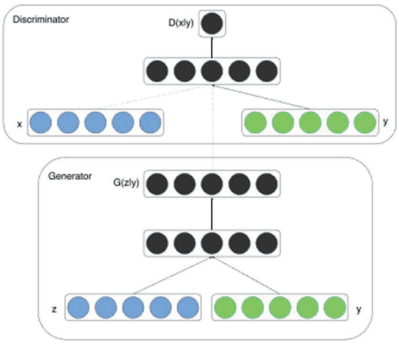

Figure 2. Simple Structure of Conditional Adversarial Net (adapted from Figure 1 in [Mehdi Mirza et al, 2014])

2.3 Conditional DCGAN (Constructed model in this study)

Figure 1 is the structure of proposed model in this study through trial and error finding stable training between Generator and Discriminator relatively quickly. In section 4 "Result" introduces the result using the Generator in this architecture.

Explaining details in this proposed model, Generator and Discriminator have the common factor that there is additional input named the auxiliary information represented one -hot (In this study, the auxiliary information is replaced to the label information related with the kind of characters). In Generator, the random noise z which consists of the number of dimensions nz and label information *c* are merged and input to Liner layers, then proceeded to up-sampling as output G(*z*,*c*) by deconvolution. In Discriminator, c is transposed to channel representation c' since Discriminator's input is represented with channel like training sample data x or G(*z*,*c*), then merged them and extract feature maps by convolution. When proceeded to convolution, input in each layer is added some noise for stable learning [Martin Arionsky et al, 2017].

Generator, Discriminator Loss is obtained by using the output from Discriminator like GAN and DCGAN. Formula (3) is loss function for Discriminator and (4) is the one for Generator.

$$L_D(G,D) = E\big[\log\big(D(x,c)\big)\big] + E[1 - \log(D(G(z,c),c))] \quad (3)$$

$$L_G(G,D) = E[-\log(D(G(z,c),c))] \quad (4)$$

3. Experiment

3.1 Handwritten character dataset

As the target for handwritten character dataset, we used ETL-1 Character Database [Electrotechnical Laboratory, 1973-1984] from Electrotechnical Laboratory (succeeding organization: National Institute of Advanced Industrial Science and Technology).

In the ETL-1 Character Database, the handwritten character images are grayscale and have a unified size of 64×63. The dataset contains 96 different characters: 10 Arabic numerals, 26 large alphabets, 12 special characters and 48 katakana letters. These handwritten characters were collected from 1445 writers, by making each writer write one character at a time on an OCR sheet. The total number of samples collected were 141,319.

In the training process of the Generator and the Discriminator, we treated this dataset as training sample data x.

3.2 Experiment Environment

The training of the Generator and Discriminator to distinguish 96 different kinds of characters is implemented in the Python programming language and Chainer deep learning library [Seiya Tokui et al, 2015]. We also used NVIDIA GeForce GTX 1080 Ti graphic boards to speed up the training as much as possible.

3.3 Experiment Setup

As an initial setting, the whole training sample images are resized to 64×64 and set the weight decay parameter λ= 0.00001.

The following steps count as 1 epoch. We repeated training the Generator and Discriminator for 100 epochs, every time employing a minibatch size 50.

Step 1:
This step consists in preparing the Generator's input, random noise and the label information. Random noise *z* is generated from uniform random distribution in the range [-1, 1], setting the number of dimensions as *nz*. Label information c is represented

with one-hot vector corresponding to the ID related to each type of character. The data shape of c becomes (batch size, label num, 1).

Step 2:

The z and c inputs are merged into the Generator to generate the output data G(z, c).

Step 3:

To the Discriminator, G(z,c) as fake data is input merged with c' which is represented in channel from c (The data shape of c' becomes (batch size, label num, h, w)). Next, the training sample data x is input merged with c'.

Step 4:

From the Discriminator's output, Generator and Discriminator Loss is calculated and the relevant parameters are updated in each model. As an optimization function, we employed the Adam function [Diederik P. Kingma et al, 2014]. The Adam function parameters in the Generator and Discriminator network models which assisted stable training in our study are shown in Table 1.

Table 1: Adam function parameters

Parameters	α	β_1
Generator	0.001	0.5
Discriminator	0.0002	0.5

4. Result

Using the Generator in our Conditional DCGAN, this section introduces the generated result changing *nz* =32,64,96(corresponding to the number of character kinds), 256,1024 and 4096(same to whole image size we set).

4.1 Generating conditional handwritten characters

To make sure the Generator output handwritten characters designating c, we prepared 5 kinds of characters. Figure 3 shows each kind of handwritten character image picked up from training sample data.

Figure 3. The targets for generating (picked up from ETL-1 Character Database)

Figure 4 is the result generated by using Generators of varying nz values. In the images depicted in Figure 4, vertical axis means the output changing label information and horizontal axis means the output using different random noise z.

From the results in *nz*=32,64 and 96, we can see that there are outputs which are likely handwritten characters, but they do not reflect the label information. Most output were the handwritten character not belong to the kind in training sample data. Also, same images are generated although changing z.

On the other hand, in Generator with nz set to 256, 1024, 4096, it is possible to generate by reflecting the designation of target character type. Thus, there is no confusion between similar characters such as "8" and "S", "シ" and "ツ" which are similar in shape. Moreover, in the result of changing the random noise, it was possible to generate an image in which its peculiarity appeared rather than a similar image, such as when the character is large or small, or the thickness of the line is different. Moreover, it was able to generate distinct characters despite the size being smaller (*nz* = 256; image size: 64 × 64).

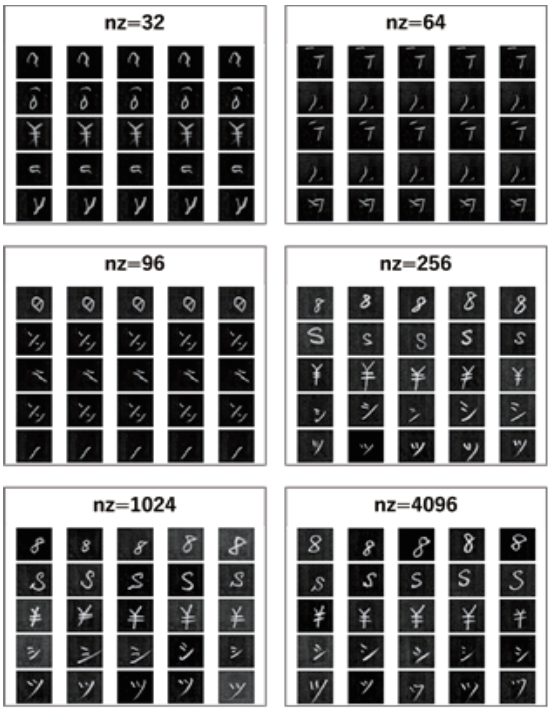

Figure 4. Conditional output from Generator changing nz

4.2 Loss changes in Generator and Discriminator

The Loss specific to the Generator and Discriminator for each epoch is shown in Figure 5.

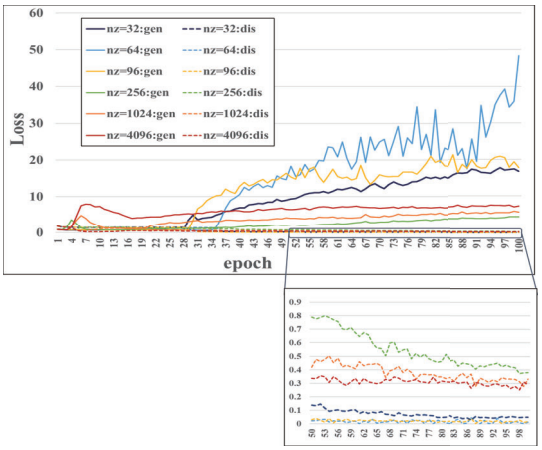

Figure 5. Loss changes from epoch 1-100.

As can be seen in Figure 5, in the model *nz* = 32,64,96, as the epochs progressed, the Generator Loss steadily increased, while the Discriminator loss gradually decreased near to 0. The difference in loss between Generator and Discriminator at epoch wider than the ones in conditional image generation. This result implies that gradient vanishing occurred in the Generator since Discriminator learned to distinguish between the real data and the fake data much before the Generator optimized to deceive the matured Discriminator [Ian Goodfellow, 2016].

Figure 6 shows the result of generating the whole of 96 kinds of characters as the target we set, from a larger *nz*=4096 to a smaller *nz*=256. Each Generator could output almost all kinds of handwritten images, making distinction just by changing the label information.

It can be inferred from our results that when the number of dimensions of noise falls below the number of labels, the model cannot generate images that are likely to be characters; on the other hand, if the number of types exceeds 90, the model can generate the specified characters.

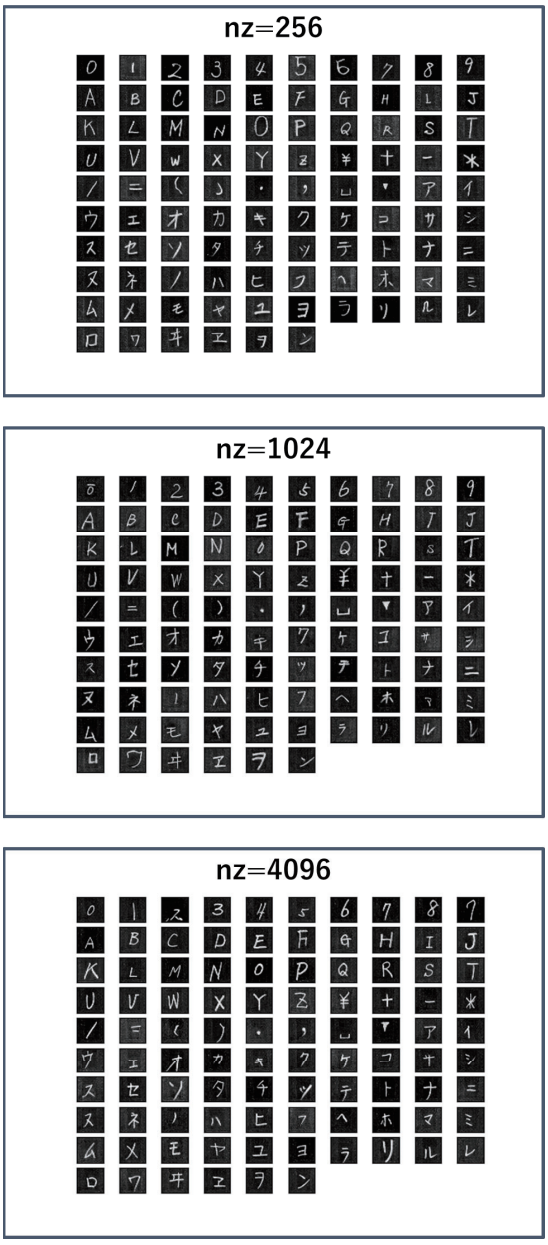

Figure 6. Handwritten characters produced by Generator

5. Conclusion and Future work

In this study, to be able to generate handwritten characters distinguishing among 96 different kinds of characters by adding UI designation, we constructed Conditional DCGAN. This model adapted DCGAN techniques using deconvolution for up-sampling at the Generator and convolution for extracting feature maps, and cGAN technique that adds label information to Generator and Discriminator. Through training our Generator and Discriminator with the dimension of random noise over the kinds, Generator could output the entire set of characters as a result.

In our future work, since the data shape of label information at the Discriminator in Figure 1 is (batch size, label num, h, w), large load will be applied to the model if dealing with over thousand kinds of characters like kanji. To solve this problem, constructing more compact Discriminator so that Discriminator's label information could keep the shape same as the one generated by the Generator and compressed through linear function. We want to try generating conditional images making distinction among over thousand kinds of images with compact Conditional DCGAN as the next challenge.

References

[Ian J. Goodfellow et al, 2014] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville & Yoshua Bengio: Generative Adversarial Nets, Advances in neural information processing systems, pp. 2672-2680, 2014.

[Naoki Shimada et al, 2017] Naoki Shimada & Takeshi Ooura: INTRODUCTION TO DEEP LEARNING WITH Chainer, Gijutsu-Hyohron Co (Japan), 2017.

[Alec Radford et al, 2015] Alec Radford, Luke Metz & Soumith Chintala: Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks, arXiv preprint arXiv:1511.06434, 2015.

[Sergey Ioffe et al, 2015] Sergey Ioffe & Christian Szegedy: Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, arXiv preprint arXiv:1502.03167, 2015.

[Mehdi Mirza et al, 2014] Mehdi Mirza & Simon Osindero: Conditional Generative Adversarial Nets, arXiv preprint arXiv:1411.1784, 2014.

[Martin Arjovsky et al, 2017] Martin Arjovsky & Léon Bottou: Towards Principled Methods for Training Generative Adversarial Networks, arXiv preprint arXiv:1701.04862, 2017.

[Electrotechnical Laboratory, 1973-1984] Electrotechnical Laboratory: Japanese Technical Committee for Optical Character Recognition, ETL Character Database, 1973-1984.

[Seiya Tokui et al, 2015] Seiya Tokui, Kenta Oono, Shohei Hido & Justin Clayton: Chainer: a Next-Generation Open Source Framework for Deep Learning, Proceedings of workshop on machine learning systems (LearningSys) in the twenty-ninth annual conference on neural information processing systems (NIPS). Vol. 5, pp. 1-6,2015.

[Diederik P. Kingma, 2014] Diederik P. Kingma & Jimmy Lei Ba: Adam: A Method for Stochastic Optimization, arXiv preprint arXiv:1412.6980, 2014.

[Ian Goodfellow, 2016] Ian Goodfellow: NIPS 2016 Tutorial: Generative Adversarial Networks, arXiv preprint arXiv:1701.00160, 2016.