Rewrite2: A GAN based Chinese font transfer method

Bo Chang
81678147
Department of Statistics
University of British Columbia
bchang@stat.ubc.ca

Shenyi Pan
83923152
Department of Statistics
University of British Columbia
shenyi.pan@stat.ubc.ca

Qiong Zhang
85896158
Department of Statistics
University of British Columbia
qiong.zhang@stat.ubc.ca

Abstract

Designing a new Chinese font is time consuming and troublesome. It can be viewed as a style transfer problem. Recently, neural network-based style transfer methods have been proposed. Specifically, Generative Adversarial Networks (GANs) are widely used in image translation. In this paper, we discuss the weaknesses of existing GAN-based methods when applied on font style transfer. Furthermore, an improved method called Rewrite2 is proposed.

1 Introduction

When creating a font in English, only 26 letters, corresponding capital letters and 10 digits need to be designed. However, this is not the case when creating Chinese fonts. Chinese has more than 80,000 characters that are totally different from each other and have different structures. Figure 1 shows the Chinese character "de" in 6 different fonts. Currently, when creating Chinese fonts, designers manually design the new look for over 26,000 Chinese characters in the GBK (a standardized character set). The design process is always time consuming. More importantly, there are always a lot of characters that are not available in a particular font. Therefore, this motivated us to use generative models to create characters in a particular font. Ideally, we can let the designers manually design a small proportion of characters in a particular font, then let the model generate the rest of the characters in the same font.

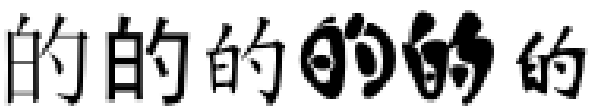


Figure 1: The Chinese character "de" in 6 different fonts: *simsun*, *simhei*, *simkai*, *hanyiheiqi*, *hanyiluobo*, *huawenxinwei* (from left to right).

To generate the rest of the characters, we have two fonts: the source font called *simsun* which is the most popular font in Chinese and has most of the characters in this format; the target font which only has a small proportion of the characters already designed.

Our Contribution: Our paper shows that the existing GAN-based methods are not "truly adversarial": they have an L_1 loss such that the generators have access to the target fonts. The existing

methods do not perform well when the L_1 loss is set to zero. To address this issues, a truly adversarial method call Rewrite2 is proposed in the paper. We also demonstrate the satisfying performance of Rewrite2.

2 Related Work

There are two approaches in the literature for solving the problems we are interested in. The first approach is to use a convolutional neural network which is trained to minimize the pixel-wise MAE (Mean Absolute Error) between predicted output and ground truth (Tian, 2017a). The second approach is based on a variant of Generative Adversarial Networks as shown in Tian (2017b).

2.1 Convolutional Neural Network and Rewrite

Rewrite (Tian, 2017a) solves the problem using a Convolutional Neural Network. Let x_i be a 160×160 image and y_i be a 80×80 image with the same character in the source and target font. 2000 characters in each font lead to 4000 images in total, which is used as the training set. The architecture of the Convolutional Neural Network (CNN) G in Rewrite is shown in Figure 2. The number of layers n is chosen to be 2, 3 or 4. The larger the n, the better the performance of the model. The loss function is defined as

$$\mathcal{L}_{CNN}(G) = \sum_{i=1}^{2000} ||G(x_i) - y_i||_1, \tag{1}$$

where the CNN G tries to minimize this objective function

$$G^* = \arg\min_G L_{CNN}(G).$$

We run the code provided by the author with number of layers $n = 2$ on a server with Nvidia

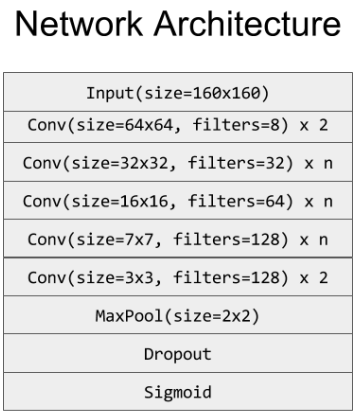


Figure 2: The Convolutional Neural Network Architecture of Rewrite.

Quadro K420, which has 2GB GPU memory. An out-of-memory error is raised after 2 iterations which shows that the approach requires high computing power even with the smallest model proposed by the author. The model pre-trained by the author also shows that the characters in the target font is blurry or even illegible when the style of the target font is hugely different from the source font *simsun*. To overcome these issues in Rewrite, a variant of Generative Adversarial Networks (GANs) is used to solve the problem.

2.2 A variant of Generative Adversarial Networks (GANs) and zi2zi

Generative Adversarial Networks (GANs) (Goodfellow et al., 2014) is an adversarial strategy where two neural network models, a generative model G and a discriminative model D, can be

trained simultaneously. The generator G is able to approximate the true distribution of the data while the discriminator D estimates the probability that a sample is from the training data rather than G. The objective function is

$$\mathcal{L}_{GAN}(G, D) = \mathbb{E}_{x\sim p_{\text{data}}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z))]. \tag{2}$$

The training procedure for G is to maximize the probability of D making a mistake. There are two major issues when training a GAN model: 1) GANs are unstable to train, resulting in generators that produce nonsensical outputs; 2) If the true distribution of the data is multi-modal, the performance of GAN is not as good as expected since we lost information of the location of the mode.

The **Deep Convolutional Generative Adversarial Networks (DCGANs)** (Radford et al., 2015) addresses the first problem by combining CNNs with GANs. In DCGANs, CNNs are used as both generators and discriminators. They essentially add a set of constraints on the architectural topology to make them stable to train in most settings. The **conditional GANs (cGANs)** (Mirza and Osindero, 2014) tries to overcome the second issue by simply feeding the labels y. We wish to condition on y for both the generator and discriminator.

One popular variant of GAN is pix2pix (Isola et al., 2016) in which a mapping from one domain X to another domain Y is approximated using a GAN model. The objective function has the form

$$\mathcal{L}(G, D) = \mathcal{L}_{cGAN}(G, D) + \lambda\mathcal{L}_1(G), \tag{3}$$

where

$$\mathcal{L}_{cGAN}(G, D) = \mathbb{E}_{x,y\sim p_{\text{data}}(x,y)}[\log D(x, y)] + \mathbb{E}_{x\sim p_{\text{data}}(x),z\sim p_z(z)}[\log(1 - D(x, G(x, z))] \tag{4}$$

and

$$\mathcal{L}_1(G) = \mathbb{E}_{x,y\sim p_{\text{data}}(x,y),z\sim p_z(z)}||y - G(x, z)||_1. \tag{5}$$

Recently, zi2zi (Tian, 2017b) uses a variant of pix2pix to generate characters in several fonts at the same time. To do that, the zi2zi model defines another loss called category loss, which tells the font that the generated character comes from. The model has an good performance when there are more than two target fonts. The form of the category loss makes the model fail to transfer the characters from *simsun* to another (single) target font. An example of the transfer from *simsun* to *simhei* can be found in Figure 3. It can be noted that in the objective function λ is set to 100 by default, which makes the L1 loss dominant. Therefore the advantage of the adversarial strategy is lost, and the model actually works similarly as Rewrite. To confirm our conjecture, we set $\lambda = 0$ in Equation 3 and use pix2pix to do the transformation from *simsun* to *simkai*. The results after 25 epoch can be found in Figure 4(a) which can be compared with the ground truth in Figure 4(b).

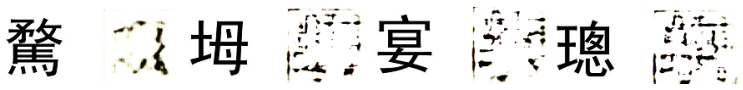


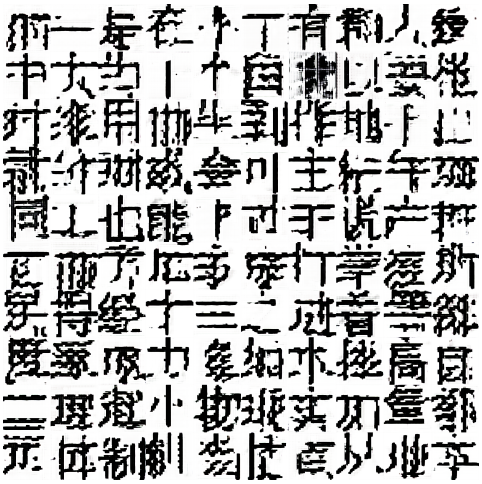
Figure 3: Four paired sample of the generated characters from zi2zi where the source font is *simsun* and the target font is *simhei*. The character on the left in each pair is the ground truth, the character on the right in each pair is the generated character.

3 Our Work: Rewrite2

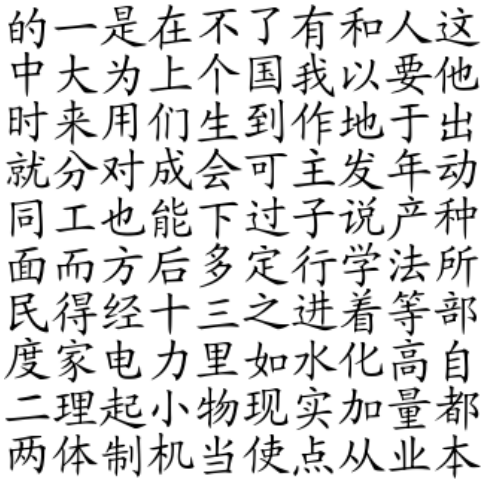
3.1 Model Specification

Our model Rewrite2 is inspired by Rewrite, GAN, DCGAN and conditional GAN. Figure 5 shows a sketch of the model architecture. Both generator and discriminator are convolutional neural networks. The model architecture is exactly the same as that in Perarnau et al. (2016). Detailed network structures are shown in Table 1 and 2.

Let x and y be 32×32 images with a same character in the source and target fonts, respectively. The input of the generator is x, and the output $G(x)$ is also a 32×32 image, which is expected to be the same character in the target font.



(a) The generated characters by pix2pix when $\lambda = 0$.



(b) The ground truth characters.

Figure 4: The comparison the generated characters and the corresponding ground truth in the test set, the characters in (a) is generated from pix2pix where $\lambda = 0$.

Operation	**Kernel**	**Stride**	**Filters**	**BN**	**Activation**
Convolution	5×5	2×2	32	Yes	LeakyReLU
Convolution	5×5	2×2	64	Yes	LeakyReLU
Convolution	5×5	2×2	128	Yes	LeakyReLU
Convolution	5×5	2×2	256	Yes	LeakyReLU
Fully connected	-	-	4096	Yes	LeakyReLU
Fully connected	-	-	100	No	None
Full convolution	4×4	2×2	512	Yes	LeakyReLU
Full convolution	4×4	2×2	256	Yes	LeakyReLU
Full convolution	4×4	2×2	128	Yes	LeakyReLU
Full convolution	4×4	2×2	64	Yes	LeakyReLU
Full convolution	4×4	2×2	1	No	Sigmoid

Table 1: Detailed generator architecture.

Our training data are paired in nature: we have each character in two different fonts. The discriminator needs to be aware of the pairing as well. Therefore, the source and target fonts are concatenated to one tensor and passed to the discriminator. Specifically, the discriminator takes (x, y) and $(x, G(x))$ as input. The output is a binary scalar: one means the input is the correct source-target pair of a character, and zero otherwise. The discriminator is optimized so that $D(x, y)$ is close to one and $(x, G(x))$ is close to zero.

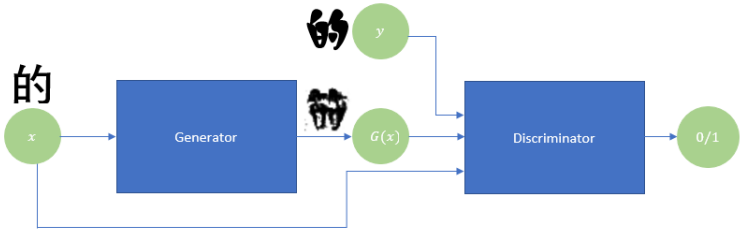


Figure 5: The model architecture of Rewrite2.

Operation	Kernel	Stride	Filters	BN	Activation
Convolution	5×5	2×2	64	No	LeakyReLU
Convolution	5×5	2×2	128	Yes	LeakyReLU
Convolution	5×5	2×2	256	Yes	LeakyReLU
Convolution	5×5	2×2	512	Yes	LeakyReLU
Fully connected	-	-	2	No	Sigmoid

Table 2: Detailed discriminator architecture.

3.2 Objective Function

The objective function of our model can be expressed as

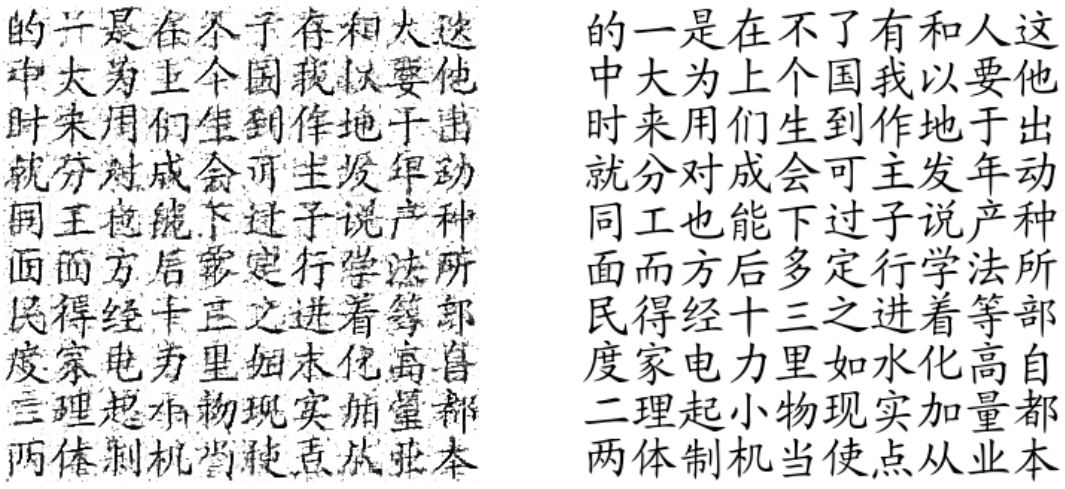
$$\mathcal{L}_{GAN}(G, D) = \mathbb{E}_{x,y\sim p_{x,y}}[\log D(x, y)] + \mathbb{E}_{x\sim p_x}[\log(1 - D(x, G(x))], \tag{6}$$

where G tries to minimize this objective against an adversarial D that tries to maximize it,

$$\min_G \max_D \mathcal{L}_{GAN}(G, D). \tag{7}$$

In pix2pix (Isola et al., 2016) and zi2zi (Tian, 2017b), the generator has a L_1 loss between $G(x)$ and y, which drastically undermines the effectiveness of the discriminator. Especially in zi2zi, our experiments show that the L_1 loss is dominant in the generator loss, which means that the effect of discriminator is negligible. Our model is a "pure" GAN model because the generator loss is not related to the target font and the its loss is determined by the discriminator.

The training algorithm is similar to that in Goodfellow et al. (2014). For each minibatch, the discriminator and generator are updated using Adam (Kingma and Ba, 2014). Our implementation[1] is based on a tensorflow implementation of DCGAN (Kim, 2017).



(a) Characters generated by Rewrite2.

(b) Ground truth characters in the test set.

Figure 6: Comparing the generated characters and the corresponding ground truth in the test set, the characters in (a) are generated by Rewrite2.

4 Experiments and Analysis

We obtained four commonly used simplified Chinese fonts: *simsun*, *simhei*, *simkai* and *huanwenxinwei*, as well as two artistic fonts: *hanyiluobo* and *hanyiheiqi*. Each contains about 3000 most frequently used Chinese characters. The characters are stored as 32×32 8-bit grayscale images. The first 100 characters are chosen as testing samples and the remaining 2900 characters are training samples. In our experiment, we use *simsun* as the source font and the others as target fonts.

[1] https://github.com/changebo/Rewrite2

The algorithm runs for 100 epochs with batch size equaling 64. The result of transferring from *simsun* to *simkai* is shown in Figure 6. See Appendix A for more experiment results. Despite some noise, most of the generated characters in Figure 6(a) are valid and legible.

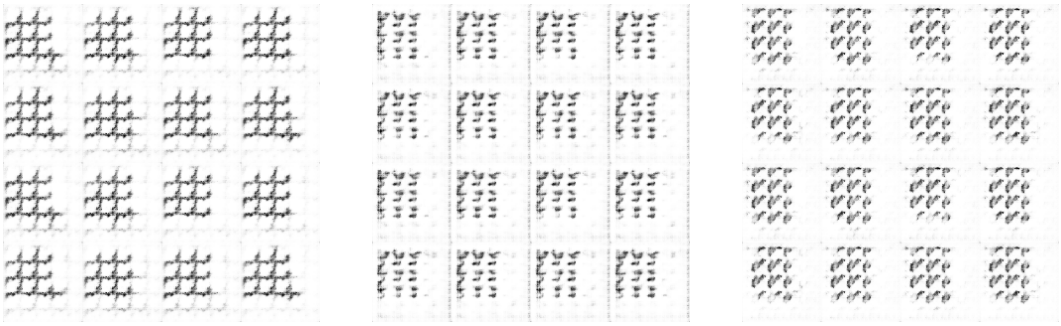
5 Discussion and Future Work

5.1 Conditional GAN: Mode Collapse

Motivated by Antipov et al. (2017), we tried another conditional GAN based approach for the purpose of transformation. In this case, the conditions y are the type of the font. For example, we have two fonts which gives the conditions of cGAN as two-dimensional one-hot vectors. The cGAN model uses the same design for the generator G and the discriminator D as in Kim (2017). Once the cGAN is trained, the transformation of the style is done is two steps:

1. Given an character image x of font type y_0, find an optimal latent vector z^* which allows to generate a reconstructed character $\bar{x} = G(z^*, y_0)$ as close as possible to the initial one x. A neural network which minimizes the pixel-wise L1 distance of ground truth and the reconstructed image can be trained to find the optimal z^*.
2. Given the target font y_{target}, generate the resulting character $x_{\text{target}} = G(z^*, y_{\text{target}})$ by simply switching the type of the font at the input of the generator.

The standard procedure for training the cGAN with 2994 pairs of *simkai* and *simhei* lead to the well-known mode collapse and the results can be found in Figure 7.



(a) After 5k iteration. (b) After 10k iteration. (c) After 15k iteration.

Figure 7: The mode collapse when train the cGAN with 2994 pairs of *simkai* and *simhei*

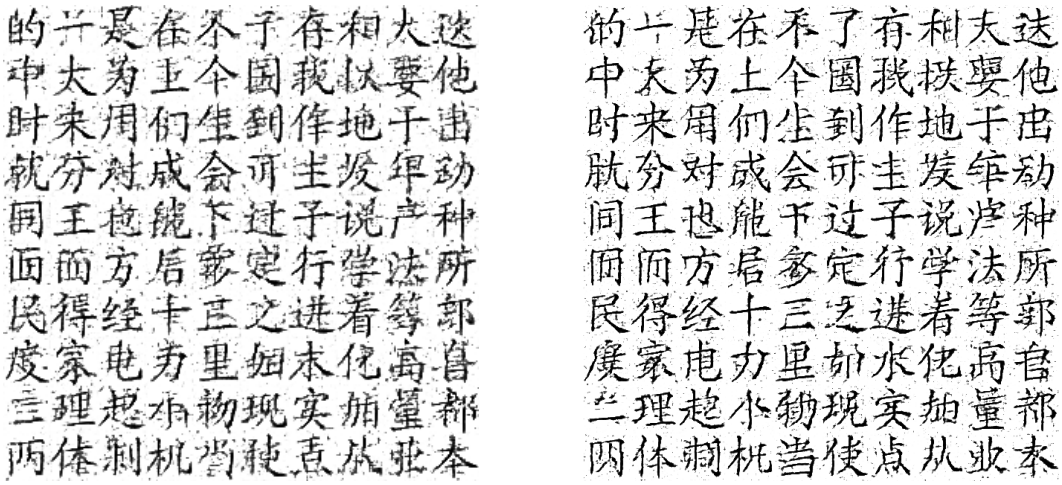
5.2 Future Work

To get rid of the noises in the generated images, we tried two approaches: 1) Use larger pictures where the edges of the characters are smoother. 2) Add a total variation loss penalty to the objective function where the total variation loss of an image x is defined as $\sum_{i,j} |x_{i,j} - x_{i-1,j}| + |x_{i,j} - x_{i,j-1}|$. The first approach can get rid of a lot of the noises as shown in Figure 8, however, the shape of some of the characters look less similar to the ground truth. The second approach does not help with denoising. We aim to investigate other denosing methods for future work so that the generated images without noise can keep the shape of the characters.

References

Antipov, G., Baccouche, M., and Dugelay, J.-L. (2017). Face aging with conditional generative adversarial networks. *arXiv preprint arXiv:1702.01983*.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.



(a) Characters generated by Rewrite2 with image size 32×32.

(b) Characters generated by Rewrite2 with image size 64×64.

Figure 8: Comparison of the characters generated by Rewrite2 with different image sizes

Isola, P., Zhu, J.-Y., Zhou, T., and Efros, A. A. (2016). Image-to-image translation with conditional adversarial networks. *arXiv preprint arXiv:1611.07004*.

Kim, T. (2017). A tensorflow implementation of "deep convolutional generative adversarial networks". `https://github.com/carpedm20/DCGAN-tensorflow`.

Kingma, D. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Mirza, M. and Osindero, S. (2014). Conditional generative adversarial nets. *arXiv preprint arXiv:1411.1784*.

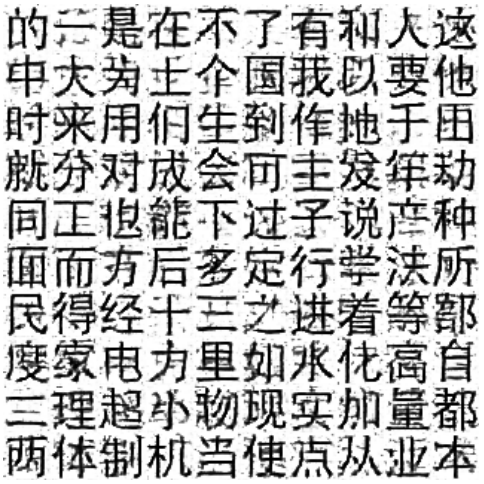
Perarnau, G., van de Weijer, J., Raducanu, B., and Álvarez, J. M. (2016). Invertible conditional gans for image editing. *arXiv preprint arXiv:1611.06355*.

Radford, A., Metz, L., and Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*.

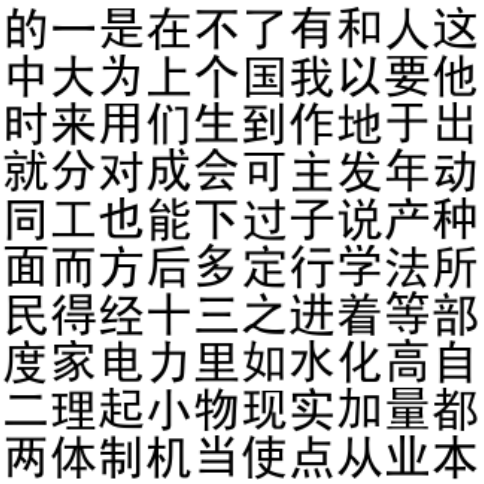
Tian, Y. (2017a). Rewrite: Neural style transfer for chinese fonts. `https://github.com/kaonashi-tyc/Rewrite`.

Tian, Y. (2017b). zi2zi: Master chinese calligraphy with conditional adversarial networks. `https://github.com/kaonashi-tyc/zi2zi`.

A More Results



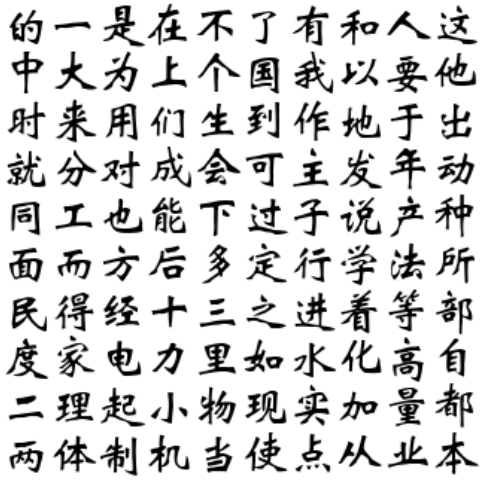
(a) Characters generated by Rewrite2.



(b) Ground truth characters in the test set.

Figure 9: Comparing the generated characters and the corresponding ground truth in the test set, the characters in (a) are generated by Rewrite2.

(a) Characters generated by Rewrite2.



(b) Ground truth characters in the test set.

Figure 10: Comparing the generated characters and the corresponding ground truth in the test set, the characters in (a) are generated by Rewrite2.

(a) Characters generated by Rewrite2.

(b) Ground truth characters in the test set.

Figure 11: Comparing the generated characters and the corresponding ground truth in the test set, the characters in (a) are generated by Rewrite2.

(a) Characters generated by Rewrite2.

(b) Ground truth characters in the test set.

Figure 12: Comparing the generated characters and the corresponding ground truth in the test set, the characters in (a) are generated by Rewrite2.