Autonomous Chinese Writing Agent

Cosmo Wang (yw259) **Kyle Zhang (guohaz)**

Abstract

Adopting an existing work on training a painting agent that paints any given image [1], our project focused on training an autonomous writing agent that can write Chinese characters stroke by stroke that resembles given pictures of such characters. Our code is available here.

1 Introduction

Writing Chinese characters is a tough task for most Chinese learners at the beginning, because each Chinese character is composed of around 10 different strokes that are arranged in different ways. Figure 1 shows the sequence of strokes needed to write one Chinese character.

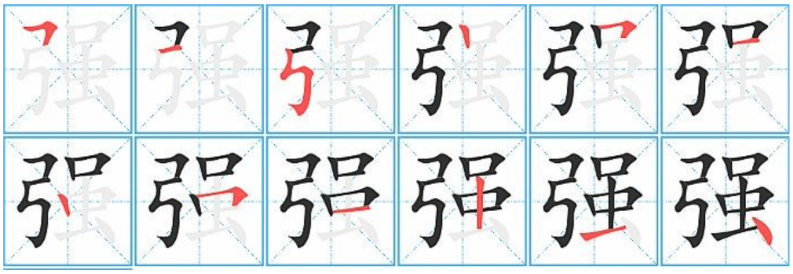

Figure 1: Correct writing process of "strong" in Chinese

Therefore, inspired by the work on autonomous painting agent [1], we attempted to modify their existing model so that we could train an autonomous writing agent that writes Chinese characters like how a human would write.

2 Adopted Work

Although painting a picture and writing a character are very different tasks, there are still a lot of commonalities. In this section, we describe the important parts from the original paper that we adopted to be further modified so that it better serves for our specific task.

2.1 The MDP Model

The ultimate goal of our writing agent is that, given a target image of a character I and an empty canvas C_0, the agent should be able to find a sequence of strokes $(a_0, a_1, \ldots, a_{n-1})$, so that after applying all those strokes in sequence, the resulting canvas C_t is visually similar to I. Here is how the task is formalized as a Markov Decision Process:

2.1.1 State

At each time step t, the state is composed of three components, the target image I, the current canvas C_t and the current step number t. So the state is formally represented as $s_t = (I, C_t, t)$. By observing the state at each step, the agent is able to learn what the current painted canvas looks like, what the number of remaining steps is and then make a decision on what stroke should be picked at this step.

2.1.2 Action

The action to be applied at each step is, intuitively, a stroke to be painted on a canvas. Formally, this action is defined as a quadratic Bézierr curve (QBC) as follows:

$$a_t = (x_0, y_0, x_1, y_1, x_2, y_2, r_0, t_0, r_1, t_1, R, G, B)_t$$

where $(x_0, y_0, x_1, y_1, x_2, y_2)$ are the coordinates of the three control points of the QBC. $(r0, t0), (r1, t1)$ control the thickness and transparency of the two endpoints of the curve, respectively.

2.1.3 Transition Function

The transition function $\mathcal{T}$ is responsible of painting a given stroke on the current canvas at each step. To accomplish this task, a neural renderer is implemented in [1]. The neural renderer is a convolutional neural network that takes in as input a stroke and outputs a new canvas with just the stroke rendered on the canvas.

Therefore with this neural renderer, at each step t of this MDP, the renderer takes in a selected action, generate a new canvas with just this action and finally stack the new canvas on C_t to get an updated canvas C_{t+1}. After this transition, we get a new state $s_{t+1} = (I, C_{t+1}, t + 1)$.

2.1.4 Reward

The reward function defined in [1] is as follows:

$$r(s_t, a_t) = L_t - L_{t+1}$$

where L_t is the measures loss between I and C_t. This loss could be any common way of measuring loss, such as an MSE loss.

With this design of the reward function, at each step, the agent is encouraged to pick an action that reduces the loss the most. In other works, the action picked at each step should drive the loss between the current canvas and the target image down as much as possible.

2.2 Model-based DDPG

The method to train such a painting agent in [1] is termed as a model-based deep deterministic policy gradient (DDPG).

In the original model-free DDPG, the actor-critic paradigm is used such that the actor $\pi(s)$ models a policy π that maps a state s_t to an action a_t and the critic estimates the expected reward of taking action a_t, which is $Q(s_t, a_t)$. Then the agent is trained using Bellamn equation and with data sampled from an experience replay buffer:

$$Q(s_t, a_t) = r(s_t, a_t) + \gamma Q(s_{t+1}, \pi(s_{t+1}))$$

The difference of the model-based DDPG proposed in [1] resides in the critic. Instead of taking in both s_t and a_t to estimate $Q(s_t, a_t)$, the critic takes in only the new state $s_{t+1} = \mathcal{T}(s_t, a_t)$, where $\mathcal{T}$ is neural renderer, and estimates $V(s_t)$ as follows:

$$V(s_t) = r(s_t, a_t) + \gamma V(s_{t+1})$$

With this modification, the action is trained to maximize $r(s_t, a_t) + \gamma V(\mathcal{T}(s_t, a_t))$ at each step t.

3 Adaption to Character Writing

To get started on our project, we first replicated the original model using our own implementation. After that, we adjusted many parts of the project to serve for our task. In this section, we discuss the modifications we made to the model to adapt it to our character writing task.

3.1 Limitation of Original Work

The authors of [1] published their pre-trained neural renderer and actor for public to test on various images. We attempted their trained models on our task and obtained the following result:

Figure 2: Writing process of "reinforce" in Chinese (2 Chinese characters) by the trained actor in [1] using 105 and 70 strokes, respectively. Pictures given are intermediate canvases separated by 15 and 10 strokes, respectively.

One can visually see that although the final canvas on the right resembles the given target image on the left, the process through which the agent accomplishes the task is more like painting rather than writing. Moreover, the agent used an excessive amount of strokes to write these two characters. The correct ways to write the two characters uses 12 and 4 strokes, respectively. This is expected because the original agent is trained to paint pictures, not to write Chinese characters.

3.2 Problem Simplifications

Given the limitation of the original work on our task, we need to tune the provided model in many ways. But before that, we made two simplifications to this problem so that it remains in the scope of a course project.

3.2.1 Order of Strokes

In Chinese, the sequence of strokes for any character needs to be written in a particular order for the writing process to be considered as correct. For example, the sequence shown in Figure 1 is the correct way of writing that character. However, to our knowledge, there exist no clean public dataset for stroke orders of all Chinese characters. Therefore, we omitted the order requirement for training our agent. In other words, the agent can generate the strokes in any order it picks, as long as the resulting canvas is visually similar to the target character.

3.2.2 Complex Strokes

Although most strokes used to write Chinese characters are considered simple, there exist complex strokes in Chinese that can not be represented by a single QBC. Figure 3 shows some examples of simple and complex strokes in Chinese.

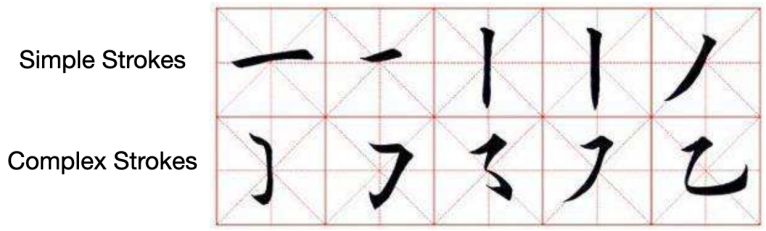

Figure 3: Example strokes in Chinese

To keep the task solvable with our computational resources, we allow the agent to write a complex stroke by composing multiple simple strokes.

3.3 Model Simplifications

Given our limited computational resources, we made some large simplifications to the original model, so it can be trained efficiently to solve our problem.

First of all, we reused the pre-trained neural renderer from [1] to save training time for the renderer. To reuse the renderer, the input to the renderer, which is a stroke described in section 2.1.2, must be kept in the same dimension in our implementation. However, the last 3 dimensions of the original stroke design represents RGB values of the stroke, which is not necessary for character writing task, because all we need for this task is a grayscale value for every pixel. So we modified the rendering code to take the average of the RGB channels of the output canvas of the renderer and treat that as grayscale value for our task.

Secondly, we reduced the dimensions of the state representation. Originally, the state was represented as $s_t = (I, C_t, t)$, where each I and C_t are all RGB pictures of dimension $3 \times 128 \times 128$. Since we only care about grayscale values, we reduced both I and C_t to have dimension $1 \times 128 \times 128$, which significantly shortened our training time for the actor and critic.

3.4 Reward Modifications

After implementing all the simplifications above, we then started to solve our core problem, which is to train an agent that writes a character, instead of painting it.

Based on our understanding, the most important difference between painting a picture and writing a Chinese character is that, the number of strokes needed for the former task is larger and each stroke is typically thicker and shorter to replicate the picture textures pixel by pixel. Therefore, we believed that we could modify the reward so that we drive the agent to use as less strokes as possible and also to prefer thin strokes.

3.4.1 Penalty for Large Step Numbers

During each training process, a maximum step number m if provided. The actor will only generate m steps in total before an update using back-propagation will happen on both the actor and the critic base on their performance.

We first attempted to encourage the agent to use less strokes, even if a large m is provided. To enforce this, on top of the original reward function described in section 2.1.4, we penalize the agent if it's approaching the maximum allowed steps. The updated reward function is as follows:

$$r(s_t, a_t) = (L_t - L_{t+1}) - \gamma_s \frac{t}{m}$$

where γ_s is a hyper-parameter to normalize the penalty with respect to the original reward.

With this penalty, as the agent is approaching the maximum allow steps, it will receive less reward even if it pick a very good stroke that decreases the measured loss by a lot. So this reward function would encourage the agent to earn as much reward as possible early on by picking long strokes.

However, after we have trained an actor using this reward function, we observed some interesting behaviors:

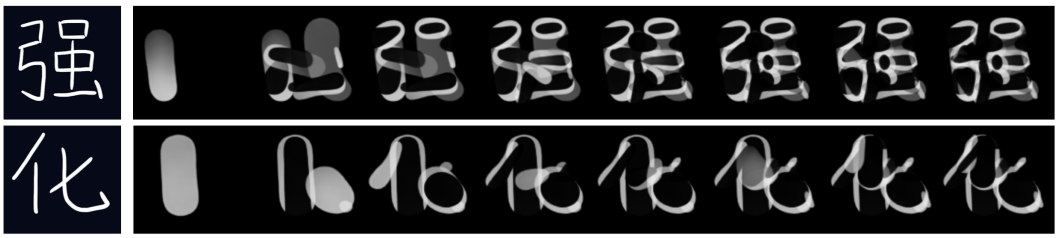

Figure 4: Writing process of "reinforce" in Chinese (2 Chinese characters) by the our actor penalized by large steps using 60 and 21 strokes, respectively. Pictures given are intermediate canvases separated by 7 and 3 strokes, respectively.

As shown in Figure 4, although the agent is able to write a visually similar characters using much less stroke in total compared the original agent in [1], it used a very non-intuitive way to accomplish this. Instead of drawing long and thin strokes, the agent tends to draw long and thick strokes and then later erase the center of the thick strokes so that it can be used to represent multiple strokes. This trick exploited by the agent successfully reduced the total number of strokes needed to write a similar character, but is not what we intended it to do.

3.4.2 Penalty for Thick Strokes

After observing the behavior of the agent in the previous section, we realized that we need to penalize the agent for cheating by using thick strokes.

Given the representation of the stroke using QBC,

$$a_t = (x_0, y_0, x_1, y_1, x_2, y_2, r_0, t_0, r_1, t_1, R, G, B)_t$$

we know that r_0 and r_1 controls the thickness of the stroke. The larger those values are, the thicker the stroke is when rendered on the canvas. Therefore, we updated the reward function to be

$$r(s_t, a_t) = (L_t - L_{t+1}) - (\gamma_s \frac{t}{m} + \gamma_t(r_0 + r_1))$$

where γ_s, γ_t are hyper-parameters to normalize the penalties with respect to the original reward.

With this penalty, the agent is encouraged to receive more reward early on in the process. And in order to do this, it can not pick too many strokes that are thick. With this reward function, we obtained the following result:

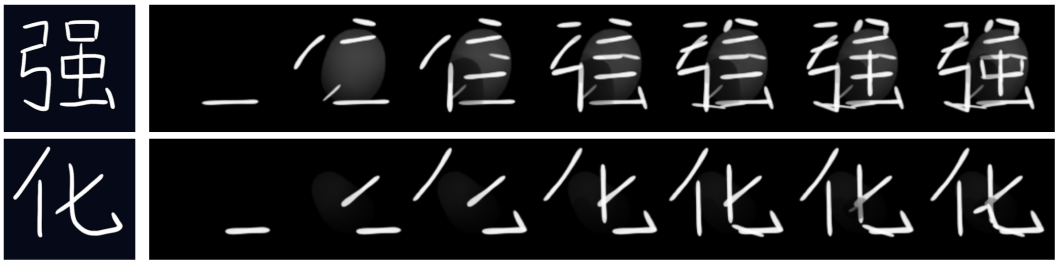

Figure 5: Writing process of "reinforce" in Chinese (2 Chinese characters) by the our actor penalized by large steps and thick strokes using 32 and 14 strokes, respectively. Pictures given are intermediate canvases separated by 5 and 2 strokes, respectively.

As shown in Figure 5, this time the agent avoided using thick strokes as much as possible and actually wrote given characters stroke by stroke. Although this is not the correct way of writing the characters in Chinese, we have achieved our goal given the simplifications we made.

4 Training and Testing

4.1 Dataset

The dataset we used to train our writing agent is the Offline Chinese Handwriting Databases collected by Institute of Automation of Chinese Academy of Sciences[2], which contains about 3.9 million samples for 7,185 Chinese characters. Due to our limited computation resources, we picked a small subset of the data to train our agent, around 240 thousand samples for 2,300 Chinese characters. Each sample is a picture of a handwritten Chinese character written on a white paper with a black pen.

Before feeding the sample to our agent, we resized all samples to have size of 128×128 and inverted the grayscale value of the sample, so the pictures have black backgrounds and white strokes. Moreover, to make the agent focus more on the strokes but not the noisy shadows in the samples, we sharpened all samples by chaning the grayscale values of all pixels larger than a threshold to 255 and smaller than the threshold to 0.

For all agents we have experimented, we trained each of them using a single NVIDIA 1060 GPU for around 40,000 mini-batches, which each took around 8 hours.

4.2 Measure for Success

One of the biggest obstacle we encountered during training is to clearly define a measure for success.

The most obvious way of measuring the performance of each agent is to take a test set, run each agent on all samples in the test set and calculate the L2 distance between the final canvas and the target image. However, we didn't think this is a good metric. Because our goal was not to recreate the target image at pixel-level, but to train an agent that mimics how a human would write those characters.

However, since we don't have large dataset of the processes of human writing Chinese characters, we could not find a quantitative way of measuring the performance of each agent. Therefore, we mostly judged each agent subjectively using a dozen of Chinese character written by ourselves. The best performing agent based on our judgement is the one trained using reward penalized by both large steps and thick strokes, with $\gamma_s = 10^{-4}$ and $\gamma_t = 2 \times 10^{-4}$. Figure 6 shows the performance of the agent on 5 Chinese characters.

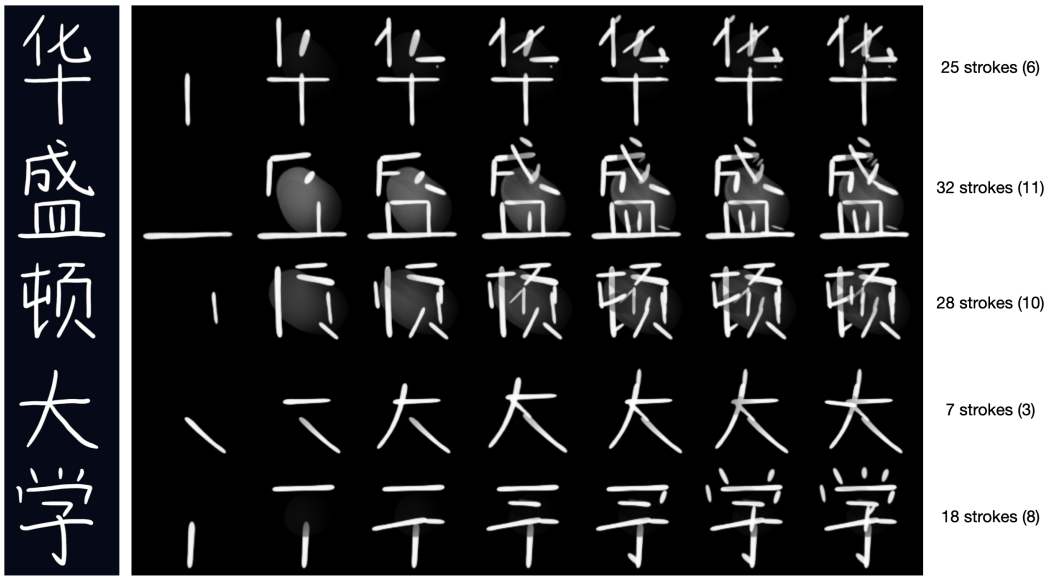

Figure 6: Writing process of "University of Washington" in Chinese (5 Chinese characters) by our best agent. n strokes (m) on the right of each row means the character is written by the agent using n strokes, while the correct number of strokes in Chinese is m.

5 Conclusion

In this project, we designed and trained a Chinese character writing agent by adopting the model for an autonomous painting agent. Upon replication of the original model, we also learned how a different reward function would drive the agent to behave differently on attempting to achieve the same task. At the end, our fine tuned agent is able to decompose a target image of a character into a few and reasonable strokes in a way that mimics how a human would write the character.

References

[1] Zhewei Huang, Wen Heng, and Shuchang Zhou. Learning to paint with model-based deep reinforcement learning. *CoRR*, abs/1903.04411, 2019.

[2] C. Liu, F. Yin, D. Wang, and Q. Wang. Casia online and offline chinese handwriting databases. In *2011 International Conference on Document Analysis and Recognition*, pages 37–41, 2011.