Learning Deep Sketch Abstraction

Yi Zhou

Abstract

Human free-hand sketches have been studied in various scenarios, including sketch recognition, synthesis and sketch-based image retrieval. One of the key challenges for sketch analysis is to deal with highly arbitrary human drawing styles, particularly in terms of abstraction level. In this context, Muhammad et al. in their work, Learning Deep Sketch Abstraction, propose a stroke-level sketch abstraction model. It is based on the observation that sketch abstraction could be deemed as a trade off between sketch recognizability and the number of strokes used to draw it. Concretely, a deep reinforcement learning agent is developed and trained to learn a stroke removal policy for abstract sketch generation, which is done by predicting which strokes can be safely removed without affecting recognizability. In this report, by strictly adhering to the methodology introduced in the reference paper, we attempt to reproduce the sketch abstraction agent and, by investigating into the implementation details, get more insight on the sketch abstraction problem.

Keywords：Sketch processing, Deep neural network, Reinforcement Learning.

1 Introduction

Sketching is an intuitive process which has been used throughout human history as a communication tool. As sketches consist of only strokes, and often only a limited number of strokes, the process of abstraction is central to sketching. A fundamental challenge in analyzing free-hand sketches is that sketches drawn by different people for the same object category/instance often differ significantly, especially in their levels of abstraction.

In this scenario, Muhammad et al. present a deep neural network for sketch abstraction, in their work, Learning Deep Sketch Abstraction. It is based on the insight that abstraction is a process of tradeoff between recognizability and brevity/compactness (number of strokes). They develop a reinforcement learning agent that learns to abstract input sketches, by estimating deep stroke saliency, thereby finding the most compact subset of input strokes, while preserving sketch recognizability. In addition, they further show several applications of their method to demonstrate effectiveness of the proposed method. In this report, we investigate into their methodology and try to reproduce the abstraction agent.

2 Related works

2.1 Sketch recognition

In sketch recognition domain, early work focused primarily on artistic drawings and computer aided design. With Yu et al. proposing the first deep convolutional neural network (CNN) designed for sketch recognition which outperformed previous hand-crafted features by a large margin, some recent work start applying learning techniques to achieve better sketch recognition result. Umar et al. exploit a sketch recognizer to

quantify sketch recognizability, and basing on this, design a recognizability-based reward scheme to train their abstraction model via reinforcement learning. On the other hand, instead of employing CNN, with more preference on sketch represented by sketch point sequence, they use a RNN-based classifier, which fully encodes stroke-level ordering information.

2.2 Sketch Abstraction

Berger et al. collected a dataset of portraits drawn by professional artists at different levels of abstraction. For each artist, a library of strokes is created indexed by shape, curvature, and length, and these are used to replace curves extracted from the image edge map. Their method is limited to portraits and requires a new dataset for each level of abstraction. Vinker et al. propose a CLIP-based object sketching method. They define a sketch as a set of Bezier curves and use a differentiable rasterizer to optimize the parameters of the curves directly with respect to a CLIP-based perceptual loss.

3 Method

3.1 Sketch representation

Sketches are represented in a vectorized format. Strokes are encoded as a sequence of coordinates, consisting of 3 elements $(\Delta x, \Delta y, p)$, as in [14] for representing human handwriting. Data-segment is defined as one coordinate and stroke-segment as a group of five consecutive coordinates. Each stroke thus comprises a variable number of stroke-segments.

3.2 Problem formulation

The sketch abstraction process is formulated as the sequence of decisions made by an abstraction agent, which observes stroke-segments in sequence and decides which to keep or remove. The agent is trained with reinforcement learning, and learns to estimate the saliency of each stroke in order to achieve its goal of compactly encoding a recognizable sketch. The RL framework is described by a standard Markov Decision Process:$\langle \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}_i \rangle$. Here: $\mathcal{S}$ is the set of all possible states, which are observed by the agent in the form of data-segments representing the sketch and the index pointing at the current stroke-segment being processed. $\mathcal{A} = 0, 1$ is the set of binary action space representing skipping (0) or keeping (1) the current stroke-segment. $\mathcal{T}(s_{t+1}|s_t, a_t)$ is the transition probability density from current state s_t to next state s_{t+1} when the agent takes an action $a_t \in \mathcal{A}$. It updates the index and the abstracted sketch so far. $\mathcal{R}(s_t, a_t, s_{t+1})$ is the reward function. At each time step t, the agent' s decision procedure is characterized by a stochastic policy $\pi_\theta = \pi(a_t|s_t, \theta)$ parametrized by θ, which represents the conditional probability of taking action at in state s_t.

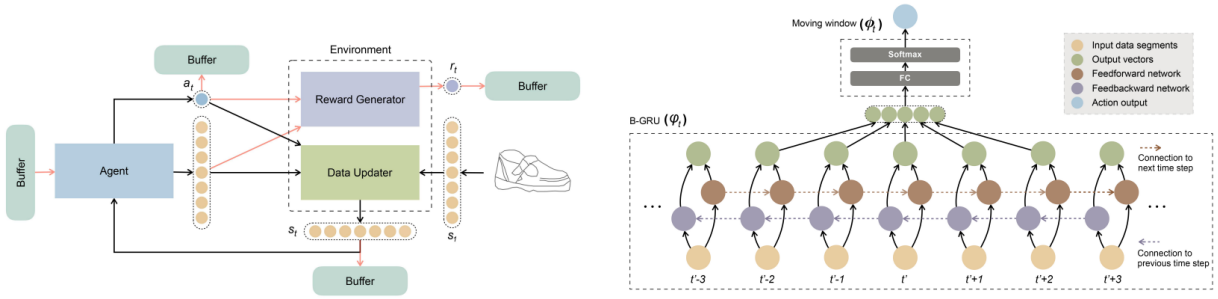
At first time step t_1, s_1 corresponds to the data-segments of the complete sketch with index pointing at the first stroke-segment. The agent evaluates s_1 and takes an action a_1 according to its policy π_θ, making a decision on whether to keep or skip the first stroke-segment. The transition $\mathcal{T}(s_2|s_1, a_1)$ says: if $a_1 = 0$ (skip), the next state s_2 corresponds to the updated data-segments which do not contain the skipped stroke-segment and with the index pointing to next stroke-segment. If $a_1 = 1$ (keep), the next state s_2 corresponds to the same data-segments as in s_1 but with the index pointing to the next stroke-segment. This goes on until the last

stroke-segment is reached. Let $D = (s_1, a_1, ..., s_M, a_M, s_{M+1})$ be a trajectory of length M, corresponding to the number of stroke-segments in a sketch. Then the goal of RL is to find the optimal policy that maximizes the expected return (cumulative reward discounted by $\gamma \in [0, 1]$):

$$J(\theta) = \mathbb{E}(\sum_{t=1}^{M} \gamma^{t-1} \mathcal{R}(s_t, a_t, s_{t+1}) | \pi_\theta) \tag{1}$$

3.3 Model

An overview of the proposed RL-based sketch abstraction model is illustrated in Fig.1.



(a) Reinforcement learning framework. (b) Agent architecture.

Figure 1: Schematic of our sketch abstraction model.

3.3.1 Agent

The agent consists of two modules. In the first B-GRU module, data-segments corresponding to state s_t are input sequentially to a recurrent neural network (RNN), i.e., one segment at each time step t_0. The second moving window module consists of a multi-layer perceptron (MLP) with two fully-connected layers. The second layer is softmax activated, and generates probabilities for agent actions ϕ_t. This module slides over the B-GRU module and takes as input those outputs centered at the current stroke-segment under processing, using the index in state s_t. The architecture of our agent is shown in Fig.1(b).

3.3.2 Environment

The environment implements state transition and reward generation. The state transition module reads (at, st) at each time step t, and transits the environment to state s_{t+1}. Given a skip action, the skipped data-segments would be eliminated from the sketch, modifying the rest appropriately given the created gap, and moving the index to the next stroke-segment. In case of a keep action, only the index pointing to the current stroke-segment is updated. The second module is a reward generator which assigns a reward to each state transition.

3.4 Reward Scheme

The reward is driven by a sketch recognizability signal deduced from the classification result of a three-layer LSTM [16] classifier, trained with cross-entropy loss and Adam optimizer [24]. With this classifier, the author proposes the following two types of reward schemes.

3.4.1 Basic reward scheme

This reward scheme is designed to encourage high recognition accuracy of the final abstracted sketch while keeping the minimum number of stroke-segments. For a trajectory of length M, the basic reward b_t at each time step t is defined as:

$$R_t = b_t = \begin{cases} +1, & \text{if } t < M \text{and } a_t = 0(\text{skip}) \\ -5, & \text{if } t < M \text{and } a_t = 1(\text{keep}) \\ +100, & \text{if } t = M \text{and Class}(s_t) = G \\ -100, & \text{if } t = M \text{and Class}(s_t) \neq G \end{cases} \tag{2}$$

where G denotes the ground truth class of the sketch, and Class(s_t) denotes the prediction of the sketch classifier on abstracted sketch in s_t. From Eq. 2, it is clear that R_t is defined to encourage compact/abstract sketch generation (positive reward for skip and negative reward for keep action), while forcing the final sketch to be still recognizable.

3.4.2 Ranked reward scheme

This scheme extends the basic reward by a more elaborate reward computation, aiming to learn the underlying saliency of stroke-segments by integrating the classification rank information at each time step t. The total reward is now defined as:

$$R_t = \omega_b b_t + \omega_r r_t \tag{3}$$

$$r_t = \begin{cases} (\omega_c c_t + \omega_v v_t) b_t, & \text{if } t < M \\ 0, & \text{if } t = M \end{cases} \tag{4}$$

$$c_t = 1 - \left(\frac{K - C_t}{K} \right) \tag{5}$$

$$v_t = 1 - \left(\frac{K - (C_t - C_{t-1})}{2 \cdot K} \right) \tag{6}$$

where r_t is the ranked reward, ω_b and ω_r are weights for the basic and ranked reward respectively, C_t is the predicted rank of ground-truth class and K is the number of sketch classes. The current ranked reward c_t prefers the ground-truth class to be highly ranked. Thus improving the rank of the ground truth is rewarded even if the classification is not yet correct. The varied ranked reward v_t is given when the ground-truth class rank improves over time steps. w_c and w_v are weights for current ranked reward and varied ranked reward respectively. For example, assuming $\omega_b = \omega_r = 0.5$, at time step t, if $a_t = 0$ (skip), then R_t would be 0.5 when $c_t = 0, v_t = 0$, and $R_t = 1.0$ when $c_t = 1, v_t = 1$; on the other hand if $a_t = 1$(keep), then R_t would be -2.5

when $c_t = 0$, $v_t = 0$, and $R_t = -5.0$ when $c_t = 1, v_t = 1$. The basic vs ranked reward weights $\omega_b \in [0, 1]$ and $\omega_r \in [0, 1](\omega_b + \omega_r = 1)$ are computed dynamically as a functions of time step t. At the first time step $t = 1$, ω_r is 0; subsequently it increases linearly to the fixed final ω_{rf} value at the last time step $t = M$. Weights ω_c and ω_v are static with fixed values, such that $\omega_c + \omega_v = 1$.

3.5 Training procedure

A policy gradient method is applied to find the optimal policy that maximizes the expected return value defined in Eq. 1. Thus the training consists of sampling the stochastic policy and adjusting the parameters □ in the direction of greater expected return via gradient ascent. In order to have a more robust training, multiple trajectories are accumulated in a Buffer B (see Fig.1(a), and update parameters of the agent every N trajectories.

4 Implementation details

4.1 Comparing with released source codes

There is neither official release nor community implementation available for this paper. The code is written from scratch. We implement the classifier and the RL agent by Tensorflow, and, as the author does, use QuickDraw[] dataset to train both the three-layer LSTM (with each layer containing 256 hidden cells) sketch classifier and the abstraction agent. Following the setup in the original paper, We use 9 categories (cat, chair, face, fire-truck, mosquito, owl, pig, purse, shoe) with 75000 sketches in each category, using 70000 for training and the rest for testing.

The B-GRU module of the agent using a single layered B-GRU with 128 hidden cells, which is trained with a learning rate η of 0.0001. The RL environment is implemented using standard step and reset functions. In particular, the step function includes the data updater and reward generator module. The classifier is trained using cross-entropy loss and Adam optimizer. The parameters of the ranked reward scheme are set to the same setup as described in the original paper: $w_{rf} = 0.5$, $w_c = 0.8$ and $w_v = 0.2$.

5 Results and analysis

The 3-layer LSTM sketch classifier achieves 90% of accuracy on the test set, 7% lower than that in the original paper. But on the other hand, albeit having followed the paper and put it all out on finding any potential bugs, unfortunately, we failed to reproduce the abstraction agent, and therefore cannot present any meaningful results, making comparison to the original paper impossible. Still, we analyse the methodology and conclude the following potential reasons:

Incorrect code. We are still not expert at reinforcement learning so we cannot promise that our code is itself bug-free. We failed to reproduce the work because of our own inadequacy.

Incorrect choice of Policy Gradient Method. Reinforcement learning is well-known to be unstable comparing to other classes of deep learning technique. Since in the paper, it is left unspecified the exact policy gradient to apply, so we instead implement the standard policy gradient, which could be deemed a bit outdated.

Sparse reward space. According to Eq. 2, reward in the last time step majors the total reward. In our training process, the agent tends to keep the fed sketch untouched so it is guaranteed to obtain the $+100$ reward

in the last time step, regardless of the -5 penalty for keeping all the stroke-segment. We've also considered training classifier and abstraction agent with different subset of training data, but since in the paper the author doesn't do so, the attempt is aborted.

6 Conclusion and future work

We studied, investigated through the paper, Learning Deep Sketch Abstraction, by Muhammad et al., and try to reproduce the core section of the paper. Although we failed on training the abstraction agent, we still learn much from the failure and find doing this task valuable, getting ourselves more familiar with the sketch abstraction problem and applying reinforcement learning to our research problems. We expect ourselves to leverage this experience in our future research, and, if possible, come back to this work and successfully reproduce the abstraction model.