SvgAI – Training Methods Analysis of Artificial Intelligent Agent to use SVG Editor

Anh H. Dang*, Wataru Kameyama**

*GITS, Waseda University, Tokyo, Japan

** Faculty of Science and Engineering, Waseda University, Tokyo, Japan

danghoanganh@akane.waseda.jp, wataru@waseda.jp

Abstract— **Deep reinforcement learning has been successfully used to train artificial intelligent (AI) agents, which outperforms humans in many tasks. The objective of this research is to train an AI agent to draw SVG images by using scalable vector graphic (SVG) editor with deep reinforcement learning, where the AI agent is to draw SVG images that are similar as much as possible to the given target raster images. In this paper, we propose framework to train the AI agent by value-function based Q-learning and policy-gradient based learning methods. With Q-learning based method, we find that it is crucial to distinguish the action space into two sets to apply a different exploration policy on each set during the training process. Evaluations show that our proposed dual ϵ-greedy exploration policy greatly stabilizes the training process and increases the accuracy of the AI agent. On the other hand, policy-gradient based training does not depend on external reward function. However, it is hard to implement especially in the environment with a large action space. To overcome this difficulty, we propose a strategy similar to the dynamic programming method to allow the agent to generate training samples by itself. In our evaluation, the highest score is archived by the agent trained by this proposed method. SVG images produced by the proposed AI agent have also superior quality compared to popular raster-to-SVG conversion softwares.**

Keywords—**Reinforcement Learning, SVG, Exploration Policy, Q-learning**

I. INTRODUCTION

Besides defeating the world best human player in Go [1], AI agent (hereafter referred as agent) trained by deep reinforcement learning (RL) [2] has achieved human-level in a wide variety of tasks like playing 3D first-person shooter game [3], and enhances the capability of robotic automation [4]. For example, Mnih et. al. have introduced deep Q-network (DQN) [5] that plays Atari 2600 games well above the skill of human players and any other linear models. Subsequently, the works on prioritized experience replay [6], double Q-network [7], duel Q-network [8] and asynchronous actor-critic method [9] further enhance the efficiency of the training process.

Manuscript received on Dec. 31, 2017. This work is a follow-up of the invited journal to the accepted & presented paper of the 20th International Conference on Advanced Communication Technology (ICACT2018).

Anh H. Dang is with Graduate School of Global Information and Telecommunication Studies, Waseda University, Tokyo, Japan (corresponding author, phone: +81-80-1367-9637, email: danghoanganh@akane.waseda.jp)

Wataru Kameyama is with School of Science and Engineering, Waseda University, Tokyo, Japan (email: wataru@waseda.jp)

On the subject of image understanding and raster-to-vector (R2V) conversion, Karpathy et. al. present a breakthrough work [10] on training a deep neural network (DNN) [11] for automatic image description. Beltramelli proposes Pix2Code DNN [12] that generates Extensible Markup Language (XML) based code from raster screenshot of graphical user interface (GUI).

Despite being a mature branch of research, image-processing based R2V conversion is not yet reliable [13]. Therefore, we propose a framework to train an agent to use SVG editor (hereafter referred as editor) with RL. The objective of this agent is to draw an SVG image that is similar as much as possible to a given target raster image. It can be considered as a new paradigm to solve the R2V problem.

In this paper, we concentrate on exploring the feasibility of this new paradigm by training the agent to work on randomly generated target images. A custom editor is created for carrying out the research, which has modeled after OpenAI Gym [14] environment due to its robustness in interface design.

We train the agent by using both Q-leaning based and policy-gradient based methods. We evaluate the agent performance by comparing the similarity between generated SVG images and target images. Finally, we compare the quality between the SVG image produced by the proposed agent with that produced by popular R2V softwares.

This paper is organized as follows: Section II describes related works on R2V and RL. Section III describes our proposal for both the agent and the environment design. Section IV describes the agent model. Then, Section V describes training and evaluation process in detail. Section VI concludes the paper with possible improvement and future works.

II. RELATED WORKS

The agent needs to be trained to ultimately convert a raster image to an equivalent vector representation by using the editor. Therefore, this research is related to not only RL but also a series of visual-vector cross model works. There are two categories of works related to this paper: image-processing based R2V and deep learning based R2V conversions.

A. Image-Processing Based R2V conversion

As mentioned in Section I, image-processing based R2V conversion is not yet reliable [13]. Major problems include difficulties of color quantization, aliasing effects, shift, superposition effects, and miss-identification of texture and text [15].

There are numerous works to try to solve the above-mentioned problems. For example, Kansal et. al. propose a framework to reproduce linear filled gradient [16]. Vector representation of halftone dots in binary images is presented by Kawamura et. al. [17]. However, they leave the work of identifying the type of problems to the human operator. Thus, for example, well-known conversion tools such as Potrace [18] still require human's intervention to achieve desirable results [19].

B. Deep Learning Based R2V conversion

Image annotation has been an active research area. It becomes overly crowded recently due to the advancement of deep learning based natural language processing and computer vision. Karpathy et. al. achieve a breakthrough with an end-to-end DNN model [10]. Learning only from images and corresponding annotations, this model not only recognizes and locates objects, but also annotates images at different hierarchical levels. The model is realized by the embedding visual model using RCNN (Regional Convolutional Neural Network) [20] and the language model using BRNN (Bi-directional Recurrent Neural Network) [21] into the same multimodal space.

Beltramelli proposes Pix2Code [12], an end-to-end DNN that generates XML based GUI code from mock-up images. Even though similar to [10] in general design, the visual model used in Pix2Code is a plain CNN block while language model is handled by a Long-Short Term Memory (LSTM) network [22] block. Another LSTM block is used to decode the network's output into code tokens. This work can be understood as a rigid version of automated image annotation. However, the model is not flexible because the visual presentations of all the GUI elements in this work are templated. Thus, it only works with GUI images based on the fixed templates.

C. Reinforcement Learning

The agent in deep RL holds a policy set that ultimately decides which action to be performed in the next step. At each state of time step t, the agent observes the state s_t of the environment, and decides action a_t based on its current policy π. Reward r_t is then given to the agent as the feedback from the environment. The objective of the training process is to train the agent so that its policy will result in maximized reward in the future. The expected reward is calculated as follows.

$$R_t = \sum_{i=t}^{T} \gamma^{i-t}\, r_\mathrm{i} \tag{1}$$

Where R_t is the sum of expected discounted reward at time step t, T is the time when the episode is terminated, and γ is the discount factor which lies within the range of [0 ... 1]. The higher the value is, the more important the future reward is.

1) Q-learning: In Q-learning, R_t is approximated by the Q function. This function returns an action-state value according to policy π as follows:

$$Q^{\pi}(s,a) = \mathbb{E}[R_t | s_t = s, a_t = a] \tag{2}$$

Hence, $Q^{\pi}(s,a)$ is essentially an expected future reward if the agent performs action a in the given state s. In practice, the action that gives the highest Q value is chosen to be executed by the agent. The optimal value $Q^*(s,a)$ is defined as follows:

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) = \max_{\pi} \mathbb{E}[R_t | s_t = s, a_t = a] \tag{3}$$

Hence, using (1) and (2), equation (3) can be written as follows:

$$Q^*(s,a) = \mathbb{E}\left[r + \gamma \max_{a'} [\![Q^*(s',a') | s,a]\!]\right] \tag{4}$$

Where s' and a' are the state and the action of the subsequent time step, respectively, and r is the immediate reward for action a given in state s. In deep Q-learning, this value is approximated by the neural network parameterized by θ:

$$Q_\theta(s,a) \approx Q^*(s,a) \tag{5}$$

Thus, with a trained agent, the Q value of an action a can be estimated as follows:

$$Q_\theta(s,a) \approx r_t + \gamma \max_{a'} Q_\theta(s',a') \tag{6}$$

Given that $y_t = r_t + \gamma \max_{a'} Q_\theta(s',a')$, the loss function is defined as follows:

$$L_t(\theta_t) = \mathbb{E}_{s,a,r,s'}\left[\left|y_t - Q_{\theta_t}(s,a)\right|^2\right] \tag{7}$$

In practice, instead of squared difference loss, Huber loss is usually used to stabilize the training process:

$$L(\alpha) = \begin{cases} \dfrac{1}{2}\alpha^2 & \text{if } |\alpha| \le \delta \\ \sigma\left(|\alpha| - \dfrac{1}{2}\delta\right) & \text{otherwise} \end{cases} \tag{8}$$

Where α is the difference between y_t and $Q_{\theta_t}(s,a)$, and δ is the point where the loss function change from quadratic to linear.

2) Policy-Gradient: Agents trained by the Q-learning method predict the state-action value as in formula (6), then the action is chosen deterministically based on this value. Thus, it heavily depends on the value function to result in better policy approximation. On the contrary, with policy-gradient based training, the agent is trained to output the action probability directly from a given state. Formally, policy-gradient optimizes policy θ to maximize the expected discount return R_t:

$$\theta = \operatorname*{argmax}_{\theta} \mathbb{E}[R_t] \tag{9}$$

To optimize the policy θ, the gradient of policy θ is given by:

$$\nabla_\theta \mathbb{E}[R_t] = \mathbb{E}[\nabla_\theta \log P(a_t) R_t] \tag{10}$$

Where $P(a_t)$ is the probability of action a at time step t. Thus, actions that lead to better expected reward R_t are encouraged. In order to train an agent using policy-gradient, R_t must be known or has to be approximated.

3) Experience Memory Replay: One of the most significant difficulties in training an agent is the strong correlation between the network policy and the action outcome of subsequent time steps. This difficulty makes online training impossible. To break the strong correlation, experience memory replay [6] is used.

In the experience memory replay, at every time step t, the experience (s_t, a_t, r_t, s_{t+1}) of the agent is stored in replay memory which is a large capacity queue. In popular works like DQN, it is usually set to store one million experiences. When it's full, the oldest experience in the queue is removed to make place for a new experience. The network is trained by using mini batches randomly drawn from this memory.

4) Exploration in RL: Training an agent using RL requires a right balance between exploitation and exploration. Exploitation is relying on the learned policy to improve the prediction accuracy while exploration allows the agent to seek for better potential solutions (i.e. avoiding sub-optimal trap). A popular exploration policy being used in RL is ϵ-greedy [2]. Under this policy, the output of the agent has an ϵ chance of being random.

There are other exploration policies based on randomization, such as Thompson sampling [23] and Bayesian sampling [24]. However, a variance of ϵ-greedy policy named reducing ϵ-greedy is most commonly used. With this policy, the agent starts with high exploration rate, and gradually reduces it throughout the training process. Thus, it allows the agent to explore in the beginning, and to focus more on exploitation in the later phase of the training process. In this paper, we evaluate the reducing ϵ-greedy variances only. All ϵ-greedy policies mentioned hereafter refer to reducing ϵ-greedy exploration policy.

III. SVG Editor

As seen in common in AI settings, the proposed framework consists of two parts: the agent and the environment. As shown in Fig. 1, the editor is playing the role of the environment in this research. For every time step, the agent observes the state of the editor (step 1). Then, the agent processes the observed state and sends a new action to the editor (step 2). So, the editor executes the action as requested and sends reward back to the agent (step 3). And the process goes back to step 1.

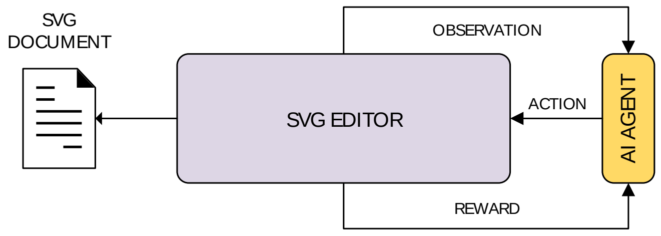

Fig. 1. The proposed framework of SvgAI. The editor is playing the role of the environment.

A. SVG Construction

SVG is an XML based vector image format. An SVG document is composed of different types of elements, such as path, circle, and rectangle. Previous works on R2V conversion usually produce SVG image using path element exclusively because this element is flexible and can be used to form any type of shape. Furthermore, using fundamental shape elements, such as rectangle, circle and arc, not only

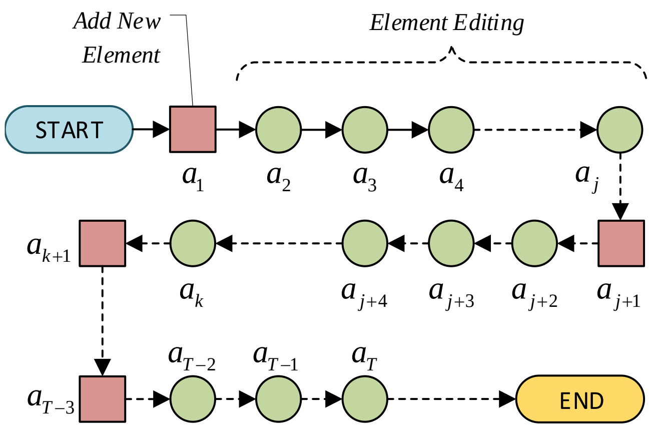

Fig. 2. The process of SVG composition: the agent keeps adding and editing new elements until the desired result is achieved. a_i is action given by the agent at time step t. T is the last time step in the episode.

requires the higher level of visual understanding but also comes with none trivial challenges.

On the other hand, using a large number of elements and paths significantly increases the SVG image size. Thus, the output SVG requires much more computational resource to be displayed. In this research, we utilize two fundamental shape elements (and with their transformations) that are square element and circle element for SVG image construction.

Fig. 2 shows the process of SVG composition, where every episode starts with a blank canvas. During the process, the agent is either adding a new element into the working image or editing the most recently added element.

The newly added element has a default presentation when it's firstly added. The default presentations of the circle and square elements are shown in sub-figure (a) and (g) of Fig. 5. As the agent keeps editing, the presentation of the element is to be updated. For example, subfigure (l) in Fig. 5 is the presentation of a circle element after 400 steps of editing.

B. Action Space

With the above-explained process, once a new element is added to the working SVG document, the old element is no longer editable. Thus, the consequence of adding new elements is more significant compared to editing them. Therefore, for Q-learning based training, it is crucial to distinguish the two sets of actions, set A and set B, and to apply separated exploration policies during the training process. Otherwise, it is impossible for the model to converge. Table I lists all the actions of set A and B supported by the editor. Set A consists of actions that add new element into the working SVG document, while set B consists of editing actions, i.e. element-shape manipulation actions.

TABLE I
List of Actions Supported By the Editor

Set	Action Id.	Description
A	0, 1	Add circle/square element
B	2, 3, 4, 5	Move element left/right/up/down
	6, 7, 8, 9	Compress/expand element horizontally/vertically
	10, 11	Rotate the element clockwise/counter clockwise
	12, 13	Reduce/increase element's line thickness
	14, .., 21	Adjust[a] element's line color (RGBα)
	22, .., 29	Adjust[a] element's fill color (RGBα)

[a] Increase/decrease value of each channel

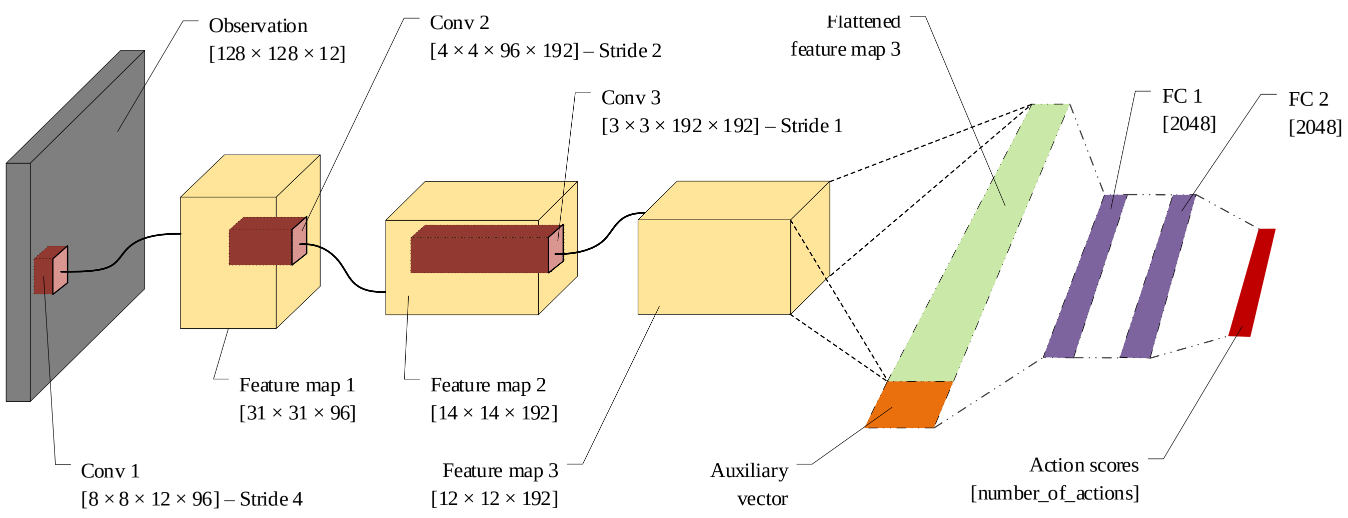

Fig. 3. The architechture of the agent. The stacked images explained in Secion III-C is processed by the convolution layers. Auxilary vector is concatenated with the output of the convolution layers before feeding to fully connected blocks at the end of the network.

C. State Observation

The state of the editor at any time step t consists of four components:

1. Raster version of the image being edited I_t.
2. Raster target image Y, i.e. the image which the agent attempts to compose by using the editor.
3. Raster image of the element being editted in I_t (component 1). Thus, , this component is exactly the same to I_t when only one element is in I_t.
4. An auxiliary vector that describes the state of the SVG element being edited (component 3). This auxiliary vector consist of paremeters such as orientation, position, line thickness, line color, and fill color.

All raster images are in the RGB α format with the resolution of 128 by 128 pixels. The first three component are stacked in channels dimention to form the image stack.

IV. Model

A. Network Architecture

Fig. 3 shows the architecture of the proposed agent. The above mentioned image stack is processed by the convolutional neural network (CNN) blocks. These blocks produce a feature map with around 27k parameters. The combination of this feature map and the auxiliary vector is then processed by a fully connected block to produce a score or probability for each action supported by the environment.

B. Error and Reward

With every action received from the agent, the score is calculated as follows:

$$g_t = \exp\left(-\frac{|I_t - Y|^2}{\sigma^2}\right) \tag{11}$$

Where g_t is the score at time step t, which describes the similarity between the rasterized version I_t of the working SVG document at time step t and the target image Y, and σ is the scaling factor. Thus, the domain of the score g_t is from 0 to 1 where 1 means perfect matching.

Base on the score g_t, the reward is given to the agent as follows:

$$r_t = \begin{cases} 1 + g_t & if\ g_t > g_{t-1} \\ 0 & otherwise \end{cases} \tag{12}$$

Where r_t is the immediate reward at time step t. Thus, the environment returns 0 reward when the action results in no improvement, and returns a small reward ranging from 1 to 2 depending on the similarity between the result and the target image.

Given v_t which is the number of SVG elements at time step t, the penalty p_t is given as follows:

$$p_t = \begin{cases} -1 & if\ v_t > v_{t-1} \\ 0 & otherwise \end{cases} \tag{13}$$

Hence, the discounted reward R_t at time step t is calculated as follows:

$$R_t = \sum_{i=t}^{T} (r_i + p_i)\gamma^{i-t} \tag{14}$$

Where γ is the discount factor and T is the length of the episode. By giving a penalty for adding elements more than necessary, the agent is discouraged to perform actions in set A. However, this technique is only feasible with a controlled environment in which the structure of a state is known. The training process is expected to be slower without this penalty.

C. Q-learning and Exploration Policy

In our early attempts, the conventional ϵ-greedy policy was used. On average, with the action sets exlained in Section III-B, there was a new element added into the image for every 14 adjustments during the exploration process. It means, regardless the value set to the exploration rate ϵ, actions in set A were explored more than necessary. This bad exploration is most likely making the network to be trapped in sub-optimal solution [25].

The simplest solution is to apply lower weight for the actions in set A during the random exploration process. The probability for an action being performed randomly by the ϵ-greedy policy is:

$$p(a_i) = \begin{cases} \epsilon\omega_a & \text{if } a_i \in \text{A} \\ \epsilon\omega_b & \text{if } a_i \in \text{B} \end{cases} \tag{15}$$

Where ω_A and ω_B are weights for the actions of set A and B. However, we observe that the agent trained by using this weighted ϵ-greedy policy often inserts incorrect element into the working image. A possible explanation for this problem is that the ϵ value is already saturated when the agent does not learn yet the long-term reward of adding the right element.

Prolonging the exploration phase does not improve the result because the agent has to keep exploring more on element editing (i.e. action set B) in order to discover better solution. Our approach to solve this problem is to apply the different reducing ϵ-greedy policies (hereafter referred as dual ϵ-greedy policy) on each action set. In this setting, there are two independent reducing ϵ-greedy polices applied for each action set:

$$p(a_i) = \begin{cases} \epsilon_a \omega_a & \text{if } a_i \in \mathrm{A} \\ \epsilon_b \omega_b & \text{if } a_i \in \mathrm{B} \end{cases} \tag{16}$$

Where ϵ_A and ϵ_B are weight values of two ϵ-greedy policies for the action set A and B, respectively. In this way, the action set A and B can be independently explored.

Algorithm 1 shows the pseudo code to train the agent using Q-learning paradigm with dual ϵ-greedy policy, where the random function has several forms (analogous to C++'s function overloading). The function Rnd(A) and Rnd(B) randomly select an action in set A and B. Rnd(X | W) picks element from X with weight W. The difficulty k is the minimum number of steps to compose the target image from a blank canvas as explained in Section IV-D.

D. Policy-Gradient

As mentioned in Section II-C-2, in order to train the agent using policy-gradient, R_t has to be known. This value can be approximated for training. Thus, it makes the efficiency of the training process again depend on an external function. Another solution is to fully unroll the episodes. Monte Carlo method is commonly used for this purpose. However, this method is intractable in the environment with a large action space, as it requires an immerse computational resource.

To overcome this difficulty, we unroll the episode and train the agent reversively. As shown in Fig. 4, supposing that k is the minimum number of steps to compose the target image from a blank canvas, if an agent is trained to perform last d step from time step t, action at time step $t-1$ can be unrolled in a brute force way:

$$P(a_i \mid s_{t-1}) = \frac{\left[D\left(s_{t|a_i}, Y\right) = \mathrm{d}\right]}{\sum_j \left[D\left(s_{t|a_j}, Y\right) = \mathrm{d}\right]} \tag{17}$$

Where $P(a_i \mid s_{t-1})$ is the probability of action a_i given the state s_{t-1} at time step $t-1$, $D\left(s_{t|a_i}, Y\right)$ is the minimum number of steps for the agent to finish the episode given in the state $s_{t|a_i}$ at time step t which is the result from action a_i from the last time step.

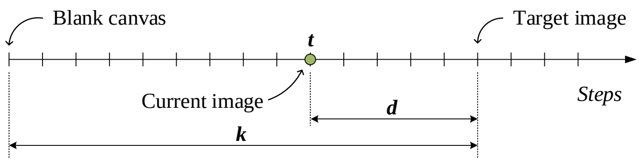

Fig. 4. The minimum number of steps to compose target image from a blank canvas is k. The minimum number of steps to compose target image from the current image is d.

Algorithm 2 shows the pseudo code of the policy-gradient based training. The agent is trained to work on the incrementally difficult states. The difficulty of a state is measured by distance d. Once the policy network θ_d for distance d is converged, it is then trained with more difficult distance $d+1$. $U(\cdot)$ is an uniform sampling function.

The primary disadvantage of this method is that it is only feasible with a controlled environment where the difficulty of the state can be calculated. However, it is useful in combination with techniques such as transfer learning when applying it for more complex data.

V. Evaluation

A. Dataset

Works on image-processing based R2V conversion have many diverse objectives. As the result, they mostly use relatively plain and small datasets. The diversity in research objectives also leads to the lack of unified evaluation dataset [15]. While the complexity of these data sets is suitable for this research, their modest size and heterogeneous properties are not adequate to be used to train DNN.

ALGORITHM 1. Q-learning Based Training Algorithm

```
inputs:    Maximum time step $t_{max}$
           Number of actions $n$
output:    Optimized policy $\theta$
variables:   $M$ experience memory
             $L$ training interval
             $t$ time step
             $gt$ global counter

while not converged
    $Y \leftarrow$ random target image $Y$ with random $k$
    for $t = 0$ to $t_{max}$
        $gt \leftarrow gt + 1$
        $a_A \leftarrow \mathrm{Rnd(A)}$ ; $a_B \leftarrow \mathrm{Rnd(B)}$
        $a_Q \leftarrow \max_a\left(Q_\theta(s_t, a)\right)$
        $a_t \leftarrow \mathrm{Rnd}\left(a_A, a_B, a_Q \mid p(a_A), p(a_B), p(a_Q)\right)$
        unroll episode with action $a_t$
        add $episode$ to $M$
        if $gt$ mod $L = 0$
            train mini batch taken from $M$
        if $episode$ end
            break
```

ALGORITHM 2. Policy-Gradient Based Training Algorithm

```
inputs:    Upper limit $k_{max}$
           Number of actions $n$
output:    Optimized policy $\theta$
variables:   $M$ experience memory
             $gt$ global counter
             $L$ training interval

for $d = 1$ to $k_{max}$
    while $\theta_d$ is not converged:
        $k \leftarrow U([d..k_{max}])$
        $t \leftarrow k - d$
        with random $episode$ at difficulty $k$
            $Y \leftarrow$ last state of $episode$
            $s_{t-1} \leftarrow$ state at time step $t-1$ of $episode$
            for $i = 0$ to $n$
                if $D\left(s_{t|a_i}, Y\right) = d$:
                    add $(s_{t-1}, a_i)$ to $M$
                    $gt \leftarrow gt + 1$
                    if $gt$ mod $L = 0$
                        train mini batch taken from $M$
```

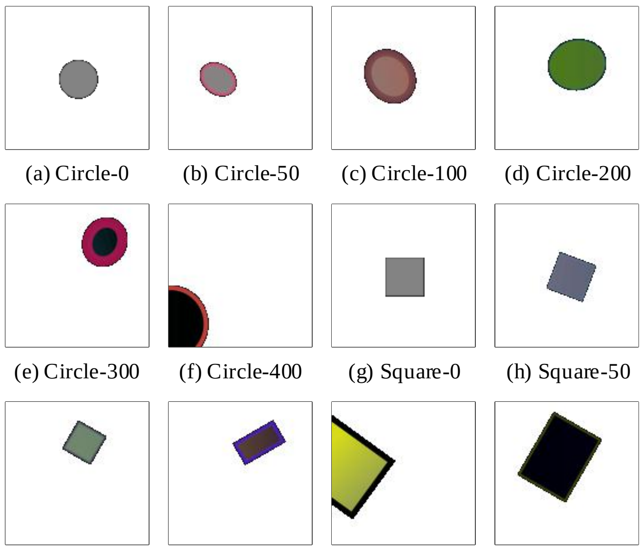

Fig. 5. Examples of images generated by training and evaluation processes. The caption under each image shows the name of the element and the minimum number of steps (k as in Fig. 4) required to compose that image from a blank canvas by using the editor.

With the above reasons, the training and the evaluation of the agent have been done on a randomly generated data set. It not only helps in avoiding the above-mentioned problems, but also provides a controlled level of difficulty and the uniformity of the dataset. Target images are created by rasterizing the randomly generated SVG documents. These documents contain a single shape element with difficulty k. Fig. 5 shows examples of target images in format <*element*>-<*k*>.

B. Episode Termination

The episode is terminated if one of the following conditions is met:

- After a fixed time step t_{max}. Since there is no further step after the termination, for Q-learning, the Q value as mentioned in (6) at the final step t_{max} is: $Q_\theta\left(s_{t_{max}}, a_{t_{max}}\right) = r_{t_{max}}$.
- When g_t in (11) is greater than a certain threshold.

C. Frame Skip

Due to the computational demand of the agent, it is inefficient to let the agent perform an action at every time step. The popular solution is the frame skip [14] where the agent only interacts with the environment in every r steps. Therefore, once an action is decided by agent, it repeats r-time steps. A popular value of r in many tasks is 4 as it is usually a good tradeoff between the performance and the training speed.

In this research, dismissing frame skip helps to improve the performance of the agent because one action repeated several times makes the agent miss its target due to overshooting.

TABLE II
HYPER-PARAMETER SETTING FOR EXPERIMENTS

		Conventional	Weighted	Dual
Minibatch size		32	32	32
Upper limit k_{max}		400	400	400
Train Interval L		16	16	16
Discount factor γ		0.999	0.999	0.999
Memory replay size		1e6	1e6	1e6
ϵ-greedy	max - min	1 – 0.1	1 – 0.1	-
	start - end	0 – 1e6	0 – 1e6	-
ϵ_A-greedy	max - min	-	-	1e-3 – 0
	start - end	-	-	5e6 – 10e
ϵ_B-greedy	max - min	-	-	0.999 – 0.
	start - end	-	-	0 – 1e6
W_A		1	1e-3	1
W_B		1	999e-3	1

D. Parameter Update

The agent does not update its parameter at every time step, but once for every L experiences added in the replay memory. Thus, given the batch size of N, one experience is learned by the agent N/L times in average.

E. Settings

We have implemented the agent using the model proposed in Section IV. We evaluate the performance of the agent trained by different training schemes:

- Policy-gradient
- Q-learning under different exploration policies
  - Conventional ϵ-greedy policy.
  - Weighted ϵ-greedy policy.
  - Dual ϵ-greedy policy as.

Table II shows the hyperparameters used for the experiments. Policy-gradient training experiments share the first 5 parameters with other experiments.

F. Performance

Fig. 6 shows the distribution of the evaluation scores of the agent under different training schemes. Similar to the training configuration, the target images used for evaluation are generated with random difficulty k given that $k \le 400$. The box plot describes the distribution of evaluation scores by 5×10^4 iterations interval from 0 to 40×10^4. For each scheme, we train the agent 5 times. Each time, we evaluate the agent after every 100 training iteration and collect evaluation score calculated using equation (11). Thus, each box describes the distribution of 500 evaluation scores. The colored box indicates IQR (interquartile range). The horizontal line within the box indicates the median score, and the extended bar indicates the maximum and minimum scores. Dots indicate outliers. Our proposed dual ϵ-greedy policy not only shows significant performance gain but also highly stable compared to conventional ϵ-greedy policy and weighted ϵ-greedy policy during the training process. The agent trained by policy-gradient achieves the best result and the high stability when the number of iterations is more than or equal to 15×10^4.

G. Accuracy

With dual ϵ-greedy policy, our trained agent favors adding circle element and achieves higher score by editing circle element in general. To further analyze this observation, we evaluate each trained agents for 500 episodes with the same setting used in Section V-F and drill down the evaluation results. In general, evaluation results can be divided into two sets:

- Circle set: consists of episodes with target images that contain circle elements only (250 episodes for each agent).
- Square set: consists of episodes with target images that contain the square elements only (250 episodes for each agent).

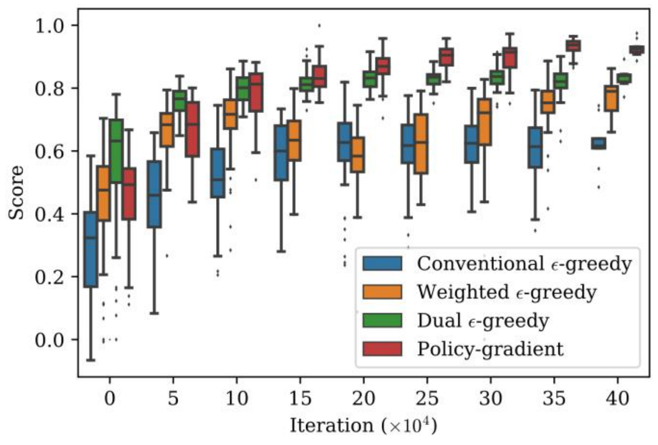

Fig. 6. Evaluation score distribution of the agent throughout the training process under different training scheme.

With each set, we analyze the performance of the agent on action set A and set B separately. The agent's performance on action set A is measured by the number of episodes where a correct shape element inserted into the working SVG document over the total number of episodes in the set. Correctly insterting a shape is important because further editing an incorrect element results in a large number of bad experience that negatively affects the agent's policy network. Table III shows the performance on action set A of the agents trained under different training schemes.

TABLE III
AGENT PERFORMANCE ON ACTION SET A

	Circle set	Square set
Policy-gradient	**0.99**	**0.95**
Q-learning		
Dual ϵ-greedy	0.94	0.76
Weighted ϵ-greedy	0.67	0.58
Conventional ϵ-greedy	0.53	0.61

To evaluate the agent's performance on action set B. We again evaluate each agent for 500 episodes with an element already inserted. The evaluation results are divided into two sets:

- Set 1: consists of episodes with target images that contain the same element with the pre-inserted element (250 episodes each agent).
- Set 2: consists of episodes with target images that contain an element that is different from the pre-inserted element (250 episodes for each agent).

Table IV shows the agent's performance on action set B. The evaluation scores of set 1 are in the grey background. For this set, the performance of agents trained by policy-gradient are better than the agents trained by Q-learning.

The evaluation scores of set 2 are in white background. Interestingly, for set 2, even with the wrong element pre-inserted, the performance of all the trained agents are over 0.5. This reflects the fact that all the agents are trained with a sustainable amount of episodes in which the wrong element is inserted. This result is also correlated to action set A performance shown in Table II: Because the agents trained by policy-gradient have high accuracy for action set A, they rarely experience the training episodes where the wrong element is added. Thus, on set 2, their performance is even lower than the performance of agents trained by dual ϵ-greedy policy (bolded italic v.s. italic on Table IV).

TABLE IV
AGENT PERFORMANCE ON ACTION SET B

Agent		Pre-insterted	Target Image	
			Circle	Square
Policy-Gradient		Circle	**0.97**	*0.65*
		Square	*0.71*	**0.94**
Q-learning	Dual ϵ-greedy	Circle	0.93	***0.78***
		Square	***0.72***	0.89
	Weighted ϵ-greedy	Circle	0.81	0.76
		Square	0.67	0.79
	Conventional ϵ-greedy	Circle	0.59	0.66
		Square	0.61	0.73

H. SVG Quality

We visually compare the SVG image produced by our trained agent with two popular opensource and commercial R2V solutions: Potrace and AutoTrace [26]. Fig. 7 shows the comparison between the outputs. As shown in the figure, Potrace not only has a problem on color quantization but also results in distorted circle. On the other hand, AutoTrace produces a much better result, however, the linear gradient fill has been converted into color blobs. Without any manual configurations prior to the conversion/drawing process, our agent produces significantly better result.

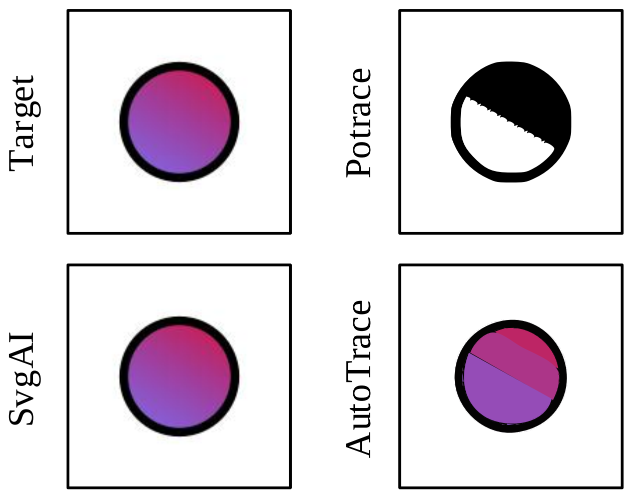

Fig. 7. Comparision between SVG images produced by our agent (SvgAI), Portrace and AutoTrace. SvgAI trained by Q-learning and policy-gradient produces identical result for this example.

Not only visually better, but the SVG images produced by our agent are also smaller both in file size and node counts. As shown in Table V, SVG images produced by our agent are 40% smaller in size and 63% smaller in node counts than the second best solution in average over 100 images where each one contains a single element, i.e. a circle or a square.

TABLE V
AVERAGE SVG SIZE COMPARISON

	Target	SvgAI	Potrace	AutoTrace
File Size	5.8KB	**852B**	1.4KB	15.5KB
Node Count	-	**1.4**	2.7	174
Color	$\geq$ 16mil	$\geq$ 16mil	2	10

VI. CONCLUSION

In this paper, we introduce a new paradigm to solve the R2V conversion problem. We propose an agent model and a framework to train the agent to use SVG editor by using Q-learning and policy-gradient. In order to successfully train the agent by using Q-learning, we divide the action space into two sets and apply independent exploration policies on each action set. Evaluation results show the efficiency of the proposed dual ϵ-greedy policy and policy-gradient. The SVG image quality produced by our agent is also superior compared to the popular software solutions. The problem of incorrect shape detection as explained in Section V-G shows a weakness of Q-learning in applying it to this problem. While policy-gradient produces the most efficient agent, it can only be used for managed environments. For future works, we will investigate and improve the training process for better reward back-propagation and for better shape detection accuracy.

REFERENCES

[1] D. Silver *et al.*, “Mastering the game of Go with deep neural networks and tree search,” *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016.
[2] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*, vol. 1, no. 1. MIT press Cambridge, 1998.
[3] G. Lample and D. S. Chaplot, “Playing FPS Games with Deep Reinforcement Learning.,” in *AAAI*, 2017, pp. 2140–2146.
[4] Lin and Long-Ji, “Reinforcement learning for robots using neural networks.” Carnegie Mellon University, 1992.
[5] V. Mnih *et al.*, “Human-level control through deep reinforcement learning.,” *Nature*, vol. 518, no. 7540, pp. 529–33, Feb. 2015.
[6] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, “Prioritized Experience Replay,” Nov. 2015.
[7] H. van Hasselt, A. Guez, and D. Silver, “Deep reinforcement learning with double Q-Learning,” *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*. AAAI Press, pp. 2094–2100, 2016.
[8] Z. Wang, T. Schaul, M. Hessel, H. Van Hasselt, M. Lanctot, and N. De Freitas, “Dueling network architectures for deep reinforcement learning,” *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*. JMLR.org, pp. 1995–2003, 2016.
[9] V. Mnih *et al.*, “Asynchronous Methods for Deep Reinforcement Learning,” in *Proceedings of The 33rd International Conference on Machine Learning*, 2016, vol. 48, pp. 1928–1937.
[10] A. Karpathy and L. Fei-Fei, “Deep Visual-Semantic Alignments for Generating Image Descriptions,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 4, pp. 664–676, Apr. 2017.
[11] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” *Commun. ACM*, vol. 60, no. 6, pp. 84–90, May 2017.
[12] T. Beltramelli, “pix2code: Generating Code from a Graphical User Interface Screenshot,” May 2017.
[13] H. S. M. Al-Khaffaf, A. Z. Talib, and R. A. Salam, “Empirical performance evaluation of raster-to-vector conversion methods: A study on multi-level interactions between different factors,” *IEICE Trans. Inf. Syst.*, vol. 94, no. 6, pp. 1278–1288, 2011.
[14] G. Brockman *et al.*, “OpenAI Gym,” Jun. 2016.
[15] V. Lacroix, “Raster-to-Vector Conversion: Problems and Tools Towards a Solution A Map Segmentation Application,” in *2009 Seventh International Conference on Advances in Pattern Recognition*, 2009, pp. 318–321.
[16] R. Kansal and S. Kumar, “A framework for detection of linear gradient filled regions and their reconstruction for vector graphics,” 2013.
[17] K. Kawamura, H. Watanabe, and H. Tominaga, “Vector representation of binary images containing halftone dots,” in *2004 IEEE International Conference on Multimedia and Expo (ICME) (IEEE Cat. No.04TH8763)*, pp. 335–338.
[18] P. Selinger and P. Selinger, “Potrace: a polygon-based tracing algorithm,” *IN HTTP://POTRACE.SOURCEFORGE.NET*, 2003.
[19] X. Hilaire and K. Tombre, “Robust and accurate vectorization of line drawings,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 28, no. 6, pp. 890–904, Jun. 2006.
[20] R. Girshick, “Fast R-CNN,” in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1440–1448.
[21] M. Schuster and K. K. Paliwal, “Bidirectional recurrent neural networks,” *IEEE Trans. Signal Process.*, vol. 45, no. 11, pp. 2673–2681, 1997.
[22] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.
[23] W. R. Thompson, “On the likelihood that one unknown probability exceeds another in view of the evidence of two samples,” *Biometrika*, vol. 25, no. 3/4, pp. 285–294, 1933.
[24] J. Asmuth, L. Li, M. L. Littman, A. Nouri, and D. Wingate, “A Bayesian sampling approach to exploration in reinforcement learning,” in *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, 2009, pp. 19–26.
[25] L. P. Kaelbling, M. L. Littman, and A. W. Moore, “Reinforcement learning: A survey,” *J. Artif. Intell. Res.*, vol. 4, pp. 237–285, 1996.
[26] M. Weber, “Autotrace-converts bitmap to vector graphics.” 2004.

Anh H. Dang (S’09) received his bachelor degree in business administration, information & communication technology from Ritsumeikan Asia Pacific University (Beppu, Oita, Japan) in 2010. He then received the master degree in computer science from Waseda University (Shinjuku, Tokyo, Japan) in 2012. Since 2012, he is a Ph.D. candidate at Waseda University. He is a member of IEEE, ACM, and IEICE. His research interests are machine learning, artificial intelligence, and computer vision.

Prof. Wataru Kameyama (M’86) received the bachelor’s, master’s, and D.Eng. degrees from the School of Science and Engineering, Waseda University, in 1985, 1987, and 1990, respectively. He joined ASCII Corporation in 1992, and was transferred to France Telecom CCETT from 1994 to 1996 for his secondment. After joining Waseda University as an Associate Professor in 1999, he has been a Professor with the Department of Communications and Computer Engineering, School of Fundamental Science and Engineering, Waseda University, since 2014. He has been involved in MPEG, MHEG, DAVIC, and the TV-Anytime Forum activities. He was a Chairman of ISO/IEC JTC1/SC29/WG12, and a Secretariat and Vice Chairman of the TV-Anytime Forum. He is a member of IEICE, IPSJ, ITE, IIEEJ, and ACM. He received the Best Paper Award of Niwa-Takayanagi in 2006, the Best Author Award of Niwa-Takayanagi in 2009 from the Institute of Image Information and Television Engineers, and the International Cooperation Award from the ITU Association of Japan in 2012.