Learning to generate HTML code from images with no supervisory data

Ali Davody [*1] Homa Davoudi [*1] Mihai S. Baba [1] Răzvan V. Florian [1]

1. Introduction

Transforming graphical designs into HTML code is a typical task of front-end web development which requires an expert programmer to spend a considerable amount of time, while this time can be dedicated to more creative problem-solving tasks where the machine intelligence is still far from the human intelligence. Automating HTML code generation from a design mock-up can serve as a solution to this problem. More recently, there have been several attempts on using machine learning techniques, especially deep neural networks, for automatic program synthesis. Among those, some works are devoted to the structured markup languages (Deng et al., 2016). However, there is still a long way to go before reaching to a satisfactory code generator. Here we introduce a new approach for generating HTML codes from a given web-page image which uses a reinforcement learning technique.

Most recent learning-based program synthesis methods require to be provided with programs generated by programmers as ground-truth. Following this supervised paradigm, the model, at best, will learn to produce programs similar to those in the training set, which, considering the inherent flexibility of programming, may not always be the most efficient ones. If each of two programmers write a program for solving the same problem, it is very unlikely that the source code of the two programs will be identical. The programs might be different in terms of their underlying algorithms, structure, applied operations and variable names. In our proposed model, instead of the ground-truth provided by humans, the supervision is carried on by a reward signal. Based on how the reward is designed, an agent learns to generate programs of desired characteristics by searching through all possible programs.

In addition to bypassing the supervision, the proposed RL based approach also solves a main drawback of typical supervised code generation methods where the inconsistency between training and testing modes is overlooked (Bahdanau et al., 2016). Considering the code generation task as an instance of sequence generation, a prevalent method is to train an RNN to generate tokens one by one conditioned on all tokens generated so far. Applying maximum likelihood framework for learning, these models will learn the conditional probability of a correct token given all the previous tokens as appeared in the ground truth. However, in the test run, the agent has only access to its own previous predictions but not the ground truth. This inconsistency between training and testing models becomes more problematic in processing of long sequences, like source code, due to the higher compounding of errors. In a RL framework, instead of maximizing the likelihood of the tokens being generated by that imprecise conditional probability, the agent is trained to directly maximize a desired evaluation metric, given as a reward signal. In our task, the agent is trained to generate a code that is rendered to the best match to the goal web page.

In our model, the generator is an agent that takes a web page image as an input and generates an HTML code that, being rendered in the browser, yields the most similar image - ideally the exact match-to the goal image. We employ a reinforcement learning approach where the agent learns to generate one token in each time step. To simplify the problem, we design a DSL to generate simple HTML pages, and the agent is trained to generate the DSL token. The reward signal is computed by running the generated code in the browser and comparing the rendered image with the goal image. In this way, only a browser is required to compute the reward, which makes the whole process independent of any other supervisory data such as source code training samples.

During the training, the agent writes a code for the goal web page based on its current policy, checks the reward to see if the written code gives the desired result, and modifies its policy accordingly.

During the test phase, the trained agent can generate the HTML code for an unseen image running the trained policy network. Nevertheless, a remarkable property of the proposed approach is that the agent has the chance of improving the result by adapting itself specifically to the test

[*]Equal contribution [1]Romanian Institute of Science and Technology, Cluj-Napoca, Romania. Correspondence to: Homa Davoudi <homa.davoudi@rist.ro>.

Published at the ICML workshop Neural Abstract Machines & Program Induction v2 (NAMPI) — Extended Abstract, Stockholm, Sweden, 2018. Copyright 2018 by the author(s).

image. This can be done by searching more as the proposed reward signal would still be available during the test, if there is access to the browser. With the reward being in hand, the agent can continue the learning process on the test time until it reaches to the exact match.

2. Model architecture

Given an image of the web page I_m, the goal is to generate its corresponding HTML code (a sequence of tokens: $t_1, t_2, ..., t_n$). In this way, the set of all possible actions the agent can take is the set of all possible tokens in the programming language. The state at time step t is composed of the goal image I_m and the codes predicted so far: $t_1, t_2, ..., t_n$). However, instead of using the code directly, we first render the partially generated code in the browser, capture the screenshot of the rendered webpage and use the rendered image, I_r, alongside the goal image I_m as the current state. This is to make it easier for the agent to compare the rendered image with the goal image.

The whole structure of the model is depicted in Figure 1. The model contains a "policy network" and a "value network" built on top of a unit base network, which encodes the current state to be fed into the following networks. The base network contains two Convolution Neural Networks (CNNs) to encode the visual information of the goal and rendered images. The feature vectors computed for the goal image and the rendered images are then concatenated and fed into the following multilayer feed forward neural networks, one dedicated to the policy and the other dedicated to the value function.

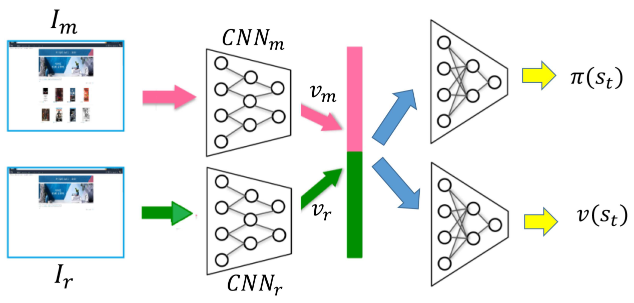
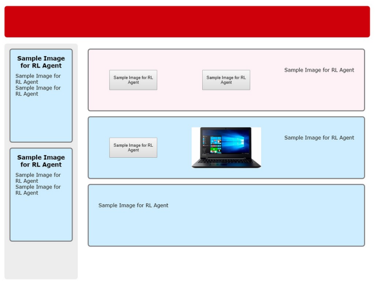


Figure 1. The architecture of our model.

To train the model parameters, the agent is provided with a reward value at each time step, which compares the rendered image up to that step with the goal image. The reward signal is computed as the negative squared L_2 distance between two feature vectors of goal and rendered images, computed in the base network using CNN networks.

These two networks are trained together through a Monte Carlo Tree Search (MCTS) algorithm (Silver et al., 2016),


Sample Image for RL Agent
Sample Image for RL Agent
Sample Image for RL Agent

(a) Web page

```
 header
left_column
{
    column_item-blue_bg
    {
        col_item_title
        col_item_text
        col_item_text
    }
    column_item-blue_bg
    {
        col_item_title
        col_item_text
        col_item_text
    }
}
right_grid
{
    grid_row-yellow_bg
    {
        grid_item_button
        grid_item_button
        grid_item_text
    }
    grid_row-blue_bg
    {
        grid_item_button
        grid_item_image
        grid_item_text
    }
    grid_row-blue_bg
    {
        grid_item_text
    }
}
```

(b) code

Figure 2. A sample from the generated data set.

to find the best policy of generating the next token in each time step given the current state. MCTS is a probabilistic approach to building a decision tree, and searching in that tree for the best action to take. As we use the pre-trained network for the CNN layers, their weights are not included among the training parameters.

When updating the parameters of the network, it should be noted that there are enough data from executing MCTS on various images. In order to make a more efficient use of data, we propose to choose the training samples more sensibly. In updating the network weights, the most appropriate data are the one gathered in a successful episode, whereas, most often, an episode ends up with an incorrect generated HTML code. To handle this, instead of using the most recent data, we use the replay buffer with a minor modification, which only preserves the data of successful episodes observed so far. This way, the policy network is not mislead by the failed experiments. This buffer is continuously updated and forms half of the updating data. The other half is filled with the recent data obtained. This trick improves the learning speed significantly.

3. Experiment

We already have conducted a number of experiments on the synthesized database built based on a DSL designed to generate simple HTML pages. The DSL has 18 tokens and the length of HTML codes can reach up to 40 tokens. An example of a webpage image along with it's corresponding code is shown in Figure 2. The work being under progress, the final results will be reported in the future publications.

Acknowledgements

This work was supported by the European Regional Development Fund and the Romanian Government through the Competitiveness Operational Programme 2014–2020, project ID P_37_679, MySMIS code 103319, contract no. 157/16.12.2016

References

Bahdanau, D., Brakel, P., Xu, K., Goyal, A., Lowe, R., Pineau, J., Courville, A., and Bengio, Y. An actor-critic algorithm for sequence prediction. *arXiv preprint arXiv:1607.07086*, 2016.

Deng, Y., Kanervisto, A., Ling, J., and Rush, A. M. Image-to-markup generation with coarse-to-fine attention. *arXiv preprint arXiv:1609.04938*, 2016.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.