EBREG-RL: Example-Based Regular Expression Generator via Reinforcement Learning

Project report for RL&IA course S25 Innopolis University

Dmitry Beresnev
AIDS-MS1, Innopolis University
d.beresnev@innopolis.university

Vsevolod Klyushev
AIDS-MS1, Innopolis University
v.klyushev@innopolis.university

Nikita Yaneev
AIDS-MS1, Innopolis University
n.yaneev@innopolis.university

1 Introduction

Nowadays big-tech companies increasingly prefer data-driven development, which requires careful extraction of information, including from unstructured or almost unstructured text. This task is considered as extremely tough, but in many real-world scenarios even unstructured text information have underlying syntactic pattern that can be characterized by a regular expressions, due to their adaptability and expressiveness. However, even for experienced programmers, writing RegEx by hand is a boring, time-consuming, and error-prone process. Moreover, despite widespread use of regular expression, there are not so many researches and literature on the automatic RegEx generation. That is why we chose an example-based RegEx generation as a topic of the project of this course.

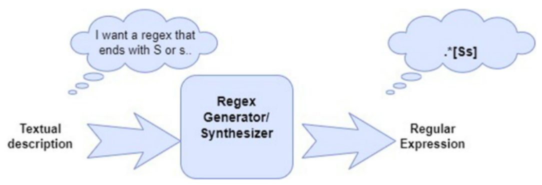
I want a regex that ends with S or s..
.*[Ss]
Textual description
Regex Generator/ Synthesizer
Regular Expression

Figure 1: NL2RE pipeline

One approach [6, 3] uses LLMs to convert natural language prompts into regular expressions (NL2RE). This method offers flexibility, allowing the model to generate a wide variety of RegExes. However, it requires significant computational resources and careful architecture design. This method is not ideal for generating RegExes that are specifically tailored to a particular domain or task. It is also challenging to ensure that the generated RegExes are accurate and efficient.

Another approach [1, 2] uses genetic programming (GP) to generate regular expression for specific tasks based on labeled data. The GP algorithm searches for the best-performing regular expression by iteratively evolving a population of candidate solutions. This method requires careful design of hyper-parameters, such as the size of the population and the mutation rate, as well as a well-defined fitness function to evaluate the performance of each regular expression.

2 Problem formulation

We propose a new approach that leverages reinforcement learning (RL) to generate regular expression. This method formulates the regular expression generation process as a Markov decision process (MDP) with fully-observable deterministic episodic environment.

Each action corresponds to a regular expression syntax unit, state space would be a set of all possible sequences of syntax units of fixed length. For reward function various information retrieval metrics can be used as indicators of how good the generated regular expression is.

2.1 Reverse Polish Notation

Due to the fact that regular expressions have operations with different precedence and number of arguments, we decided to use the Reverse Polish Notation (RPN) for the construction of regular expressions.

In our RPN implementation we have 104 different actions:

- 93 tokens (letters, digits, quantifiers, finish token, etc.),
- 2 binary operations (`concat` and `|`),
- 6 unary operations (`*`, `+`, `?`, `*?`, `+?`, `??`),
- 3 many to one operations (`[]`, `^[]`, `concat_all`).

2.2 Dataset

For the stated problem our dataset would have the following form:

- Example text
- Nonempty list of indexes of symbols from example text, which we want to retrieve with achieved regexp

3 Methodology

This section explains the basic parts of Reinforcement Learning.

3.1 Action Space

Action space for our RL agent consists of all possible actions for RPN.

3.2 State Space

The state space consists of n placeholders, where each placeholder can either contain an action from the action space or remain empty (denoted by a special empty token). To derive the current regular expression (state), we iterate over the non-empty tokens and apply the corresponding actions using Reverse Polish Notation (RPN).

3.3 Reward

Reward function is a major part of RL pipeline. We come up with 2 approaches for reward calculation. Note, that we will calculate such rewards only by the end of the episode. Before that we will give 0 as reward. Let's start with common parts.

First, we start with a text from which we want to extract specific information. To do this, we need to: Identify the indices of the characters we want to retrieve. Construct a 'target bit mask' of length n (where n is the text length), where:

- 1 marks positions to be included.
- 0 marks positions to be excluded.

Then, we get the current regular expression from RPN and apply it to the text, retrieving indices of found symbol. After that, we construct the 'current bit mask' in the same way as 'target bit mask'

3.3.1 Xor approach

First idea of the reward is the following:

$$R = \alpha \cdot \sum(A \oplus B) + \beta \cdot |C - D| + \gamma \cdot E \tag{1}$$

where:

- A — ‘target bit-mask’
- B — ‘current bit-mask’
- C — number of target words
- D — number of words, retrieved with regexp
- E — token length of regexp

3.3.2 Metrics approach

Second idea of the reward is based on the fact, that we can calculate accuracy and F1 on bit-masks. Thus reward function would be the following:

$$R = \alpha \cdot F1 + \beta \cdot A + \delta \cdot |C - D| + \omega \cdot E \tag{2}$$

where:

- F1 — F1 score on bit masks
- A — Accuracy on bit masks
- C — number of target words
- D — number of words, retrieved with regexp
- E — token length of regexp

4 RL approaches

We decided to check 3 RL approaches: Advantage Actor-Critic (A2C), REINFORCE and Deep Q-Network (DQN).

4.1 REINFORCE

REINFORCE is a version of Monte-Carlo Policy Gradient algorithm proposed by Williams in [4]. It adopts an explicit stochastic policy, and uses the episode samples to update the policy parameters. To construct the sample space, the full episode should be played, which correlates with the proposed solution architecture.

Policy In the case of using REINFORCE, the stochastic policy of selection action a_t given state s_t obtained from policy:

$$\pi_\theta(a_t|s_t) = \mathrm{softmax}(\text{MLP}(\theta)), \tag{3}$$

where θ denotes the parameters of policy neural network.

During training stage, the corresponding probabilities from eq. (3) are used for action sampling. During evaluation and testing stages, the optimal action a_t^* is chosen in the following way:

$$a_t^* = \mathrm{argmax}_a\, \pi_\theta(a|s_t) \tag{4}$$

Objective function The expected reward definition is manly based on one given in [5].

The main goal of policy optimization is maximizing the expected reward, which is defined as

$$\begin{aligned} J(\theta) = \mathbb{E}_{(s_t,a_t)\sim P_\theta(s_t,a_t)} r(s_1a_1\ldots s_Na_N) = \sum_{s_1a_1\ldots s_Na_N} & P_\theta(s_1a_1\ldots s_Na_N)R_N \\ = \sum_{s_1a_1\ldots s_Na_N} & p(s_1)\prod_t \pi_\theta(a_t|s_t)p(s_{t+1}|s_t,a_t)R_N. \end{aligned} \tag{5}$$

Here for the simplicity it is assumed, that N — total length of both target and candidate texts. By the solution architecture design, the state s_{t+1} is completely determined by state s_t and action a_t, the probabilities $p(s_1)$ and $p(s_{t+1}|s_t,a_t)$ are both equal to 1. Therefore, the eq. (5) can be simplified to

$$J(\theta) = \sum_{s_1a_1\ldots s_Na_N} \prod_t \pi_\theta(a_t|s_t)R_N. \tag{6}$$

The likelihood ratios are then used to compute the policy gradient:

$$\nabla_\theta J(\theta) = R_N \sum_{t=1}^{N} \nabla_\theta \log \pi_\theta(a_t|s_t). \tag{7}$$

In addition to just policy gradient, the entropy H of the policy is included to the objective function. Entropy is defined as

$$H(\theta) = -\sum_a \pi_\theta(a|s) \log \pi_\theta(a|s). \tag{8}$$

The value of H is always non-negative, and has a single maximum when all the actions have the same probability to be taken, i.e. when policy is uniform. Entropy reaches minimum, when $\pi_\theta(a_i|s) = 1$ for some action a_i and $\pi_\theta(a_j|s) = 0$ for all $j \neq i$. Therefore, subtracting the entropy from the objective function pushes the policy to be uniform, what means punishing the agent to be too sure about what action to take.

Finally, the objective function for the policy update combines eqs. (7) and (8), and is defined as

$$Z(\theta) = \nabla_\theta J(\theta) - \beta \nabla_\theta H(\theta), \tag{9}$$

where β is hyperparameter to balance two terms, which is usually called *entropy beta*.

4.2 Advantage Actor-Critic (A2C)

The Actor-Critic method combines the policy-based and value-based approaches. In this method, two function approximations are learned: policy function $\pi_\theta(a|s)$, which controls what action the agent selects at state s, and value function $v_\theta(s)$, which estimates how profitable the state s is.

The modification of Actor-Critic method, which uses advantage, is called Advantage Actor-Critic (A2C) and is used for more stable convergence in comparison with only policy-based methods, like REINFORCE. The advantage is calculated in the following way:

$$A(s_t, a_t) = q(s_t, a_t) - v(s_t),$$

where $q(s_t, a_t)$ — action value function.

$$q(s, a) \approx R_N = \sum_{t=1}^{N} \gamma_{\text{decay}}^{N-t} r_t. \tag{10}$$

where γ_{decay} — decay factor, which is between 0 and 1, and r_t is the reward at state s_t and total episode return R_N.

Advantage $A(s_t, a_t)$ indicates the extra reward the agent can get if it selects the action a_t at state s_t compared to the mean reward, $v(s_t)$, it gets at state s_t. Using the eq. (10), the formula for advantage becomes the following:

$$A(s_t, a_t) = R_N - v(s_t). \tag{11}$$

Policy The policy calculation is similar to REINFORCE case (eq. (3)). The stochastic policy of selection action a_t given state s_t obtained from policy head of neural network depicted on fig. 2.

During training stage, the corresponding probabilities $\pi_\theta(a_t|s_t)$ are used for action sampling. During evaluation and testing stages, the optimal action a_t^* is chosen in the same way as in eq. (4).

Objective function At the beginning, it is necessary to define $v_\theta(s_t)$. The value function of the given state s_t obtained from value head of neural network depicted on fig. 2.

The objective function itself in case of A2C consists of three parts: policy gradient part, entropy part and value loss part.

First part is policy gradient, which is quite similar to the one give in eq. (7):

$$\nabla_\theta J(\theta) = \sum_{t=1}^{N} A(s_t, a_t) \nabla_\theta \log \pi_\theta(a_t|s_t).$$

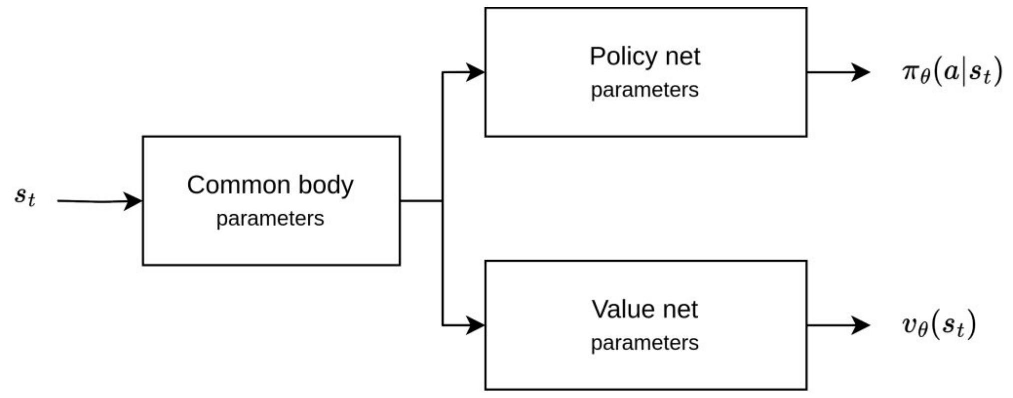

Figure 2: Advantage Actor-Critic (A2C) architecture with shared body.

The only difference is that, instead of $q_\theta(s_t, a_t)$ used eq. (7), here $A(s_t, a_t)$ is used. Using the eq. (11), the policy gradient formula becomes as following:

$$\nabla_\theta J(\theta) = \sum_{t=1}^{N} (R_N - v_\theta(s_t)) \nabla_\theta \log \pi_\theta(a_t|s_t). \tag{12}$$

Second part of objective function is entropy part, $H(\theta)$, which is identical to one proposed for REINFORCE (eq. (8)).

Third and final part of objective function is value loss part, which estimates how accurate DNet approximates value. The value loss is defined as

$$L_v(\theta) = \sum_{t=1}^{N} (q_\theta(s_t, a_t) - v_\theta(s_t))^2 = \sum_{t=1}^{N} (R_N - v_\theta(s_t))^2. \tag{13}$$

The objective function is defined as the following combination of eqs. (8), (12) and (13):

$$Z(\theta) = \nabla_\theta J(\theta) + \nabla_\theta L_v(\theta) - \beta \nabla_\theta H(\theta), \tag{14}$$

where β has the same meaning as in eq. (9).

4.3 Deep Q-Network (DQN)

The Deep Q-Network (DQN) approximates a state-value function $Q(s, a)$ for each possible action and state, using the experience replay buffer. Therefore, the DQN policy, which is build based on the state-value function, is implicit and deterministic.

Q-Network is usually optimized towards target network with frozen weights, which is denoted as $\hat{Q}(s, a)$. In its turn, the target network is periodically updated with the latest Q-Network weights every k_{sync} episodes, where k_{sync} is a hyperparameter. The target network is introduced to stabilize training.

Policy To perform agent exploration, the ϵ-greedy strategy is used. $\epsilon(i)$ is a function $\mathbb{R}^d \to (0, 1)$, which maps the index of current epoch to float value between 0 and 1. On each step, the agent with the probability of ϵ selects random action. Otherwise, the agent selects action which maximizes $Q(s, a)$. This can be formalized as follows:

$$\pi_\theta(s_t) = \begin{cases} a \sim \{a_m\} & \text{with probability } \epsilon, \\ \text{argmax}_a Q_\theta(s_t, a) & \text{with probability } 1 - \epsilon. \end{cases} \tag{15}$$

where θ denotes the parameters of DNet and $\{a_m\}$ — actions space, which is in case of this research is $\{$*Retain*, *Delete*$\}$.

During training stage, the actions are sampled according to the eq. (15). During evaluation and testing stages, the optimal action a_t^* is chosen as $a_t^* = \text{argmax}_a Q_\theta(s_t, a)$.

Objective function In case of DQN, the objective function for the episode consists only form one part — state-value loss $L_Q(\theta) = \sum_{t=1}^{N} L_{Q_t}(\theta)$, and $L_{Q_t}(\theta)$ is defined as following:

$$L_{Q_t}(\theta) = \begin{cases} (Q_\theta(s_t, a) - R_N)^2, & \text{if the episode has ended} \\ \left(Q_\theta(s_t, a) - \max_{a'} \hat{Q}_{\hat{\theta}}(s_{t+1}, a')\right)^2, & \text{otherwise.} \end{cases}$$

Note that for non-terminal states the target network $\hat{Q}(s, a)$ is used to predict value of the next state s_{t+1}. Also recall that every k_{sync} episodes $\hat{\theta}$ is set to θ.

Therefore, the objective function looks like the following:

$$Z(\theta) = \nabla_\theta L_Q(\theta). \tag{16}$$

5 Experiments

We decided to check our models on 3 different experiments.

5.1 Single number retrieval

The task is to retrieve the number from the string.

Dataset The dataset contains 500 strings of lengths from 3 to 10. Each string includes one number (maximum 3 digits).

Target Regexp We want our RL agent to simulate the application of `\d` regular expression.

5.2 Word from subset of symbols retrieval

The task is to retrieve from the string a word which consists of a certain subset of symbols.

Dataset The dataset contains 500 strings of lengths from 6 to 20. Each string includes one word that consists of the letters ‘c’, ‘a’ and ‘t’.

Target Regexp We want our RL agent to simulate the application of `[cat]+` regular expression.

5.3 Simple email retrieval

The task is to retrieve an email from the string.

Dataset The dataset contains 500 strings of lengths from 11 to 35. Each string contains one email, which consists of 5 parts: from 4 to 8 letters, `@` sign, from 3 to 4 letters, dot, from 2 to 5 symbols.

Target Regexp We want our RL agent to simulate the application of `\w+@\w+\.\w+` regular expression.

6 Results and Discussion

In this work, we managed to implement an agent who learns how to form a regular expression in reverse Polish notation. Learning on our dataset, the agent successfully learned how to get numbers and words consisting of certain letters. However, the agent cannot handle the formation of more complex expressions, for example, to get mail.

We used the DQN algorithm, but we didn’t get good results on it, because this algorithm is a value-based algorithm. Because of this, the problem of Exploration-Exploitation Trade-off arises in the algorithm, the problem of choosing ϵ is quite non-trivial. Because of this, the algorithm showed very poor convergence. Therefore, we decided to abandon it and switch to algorithms that update the policy directly.

On these graphs, you can see the results of the work for each algorithm proposed by us.

Graphs 1–2 show the results of learning algorithms for extracting numbers from a string.

Graphs 3–4 show the result of learning algorithms for extracting words consisting of certain letters.

Graphs 5–6 show the results for forming a regular expression to extract the postal address.

It can be seen from the graphs that the Reinforce algorithm is . . . , while the Actor is Critical . . . .

Our results can be improved by selecting better parameters for the algorithms, and we can also try other policy-based algorithms.

7 Conclusion

In this project our team successfully defined RL problem for regexp generation, implemented environment and tested several approaches. While some of them showed not the best performance, we proved, that such task might be solved via RL.

References

[1] Alberto Bartoli, Andrea De Lorenzo, Eric Medvet, and Fabiano Tarlao. Inference of regular expressions for text extraction from examples. *IEEE Transactions on Knowledge and Data Engineering*, 28(5):1217–1230, 2016.

[2] Alberto Bartoli, Andrea De Lorenzo, Eric Medvet, and Fabiano Tarlao. Active learning of regular expressions for entity extraction. *IEEE Transactions on Cybernetics*, 48(3):1067–1080, 2018.

[3] Sadia Tariq and Toqir Ahmad Rana. Automatic regex synthesis methods for english: a comparative analysis. *Knowledge and Information Systems*, 67(2):1013–1043, October 2024.

[4] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8(3-4):229–256, May 1992.

[5] Tianyang Zhang, Minlie Huang, and Li Zhao. Learning structured representation for text classification via reinforcement learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 32(1), April 2018.

[6] Zexuan Zhong, Jiaqi Guo, Wei Yang, Jian Peng, Tao Xie, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. Semregex: A semantics-based approach for generating regular expressions from natural language specifications. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1608–1618. Association for Computational Linguistics, 2018.