
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 have chosen an example-based RegEx generation as a topic of the project of this course.

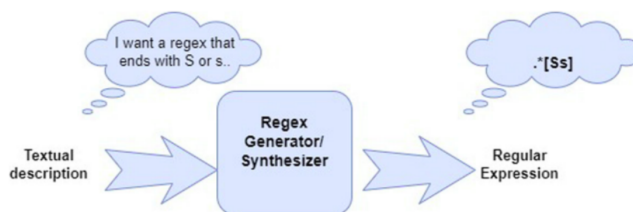


Figure 1: Natural Language to Regex (NL2RE) pipeline

One approach [6, 3] uses LLMs to convert natural language prompts into regular expressions Figure 1. 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.

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 Regex RPN implementation we have 103 different tokens:

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

Let us demonstrate, how RPN representation is converted to the infix (regular) notation. Suppose the following list of tokens as input to RPN: `[1, a, b, [], *]` By iterating one after another, the following (infix) expressions would be generated:

1. `1`
2. `a1`
3. `ba1`
4. `[ba1]`
5. `[ba1]*`

Another example with the RPN `[\d, @, |, [], +]`

1. `\d`
2. `@\d`
3. `\d|@`
4. `[\d|@]`
5. `[\d|@]+`

You can find more about RPN [here](#).

2.2 Dataset

We have decided to construct the dataset (given true regular expression) in the following form:

- Text
- Nonempty list of indexes (begin and end) of symbols from example text, which we want to retrieve with achieved regex

3 Methodology

This section explains the underlying RL theory and solution design aspects.

3.1 Action Space

Action space for our RL agent consists of all possible RPN tokens together with `FIN` state (to finish regex construction earlier). In total we have 104 actions in our action space.

3.2 State Space

The state space s 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). After the agent takes an action at step i (say we count from 0, so the first step is at $i = 0$) a_i , the state is updated as $s_i \leftarrow a_i$.

3.3 Reward

The reward in the considered task is delayed and is calculated in the final state (if the action is **FIN** or the maximum length of regex is achieved). The rewards on each non-terminate state is 0. We come up with two approaches for reward calculation. Let's start with common parts.

Firstly, we start with a text from which we want to extract specific information. To do this, we need to 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 create the 'found' regular expression from the terminate state form 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 was the following:

$$R = a \cdot \sum (A \oplus B) + b \cdot |C - D| + c \cdot E \quad (1)$$

where:

- A — 'target bit-mask'
- B — 'current bit-mask'
- C — number of target words
- D — number of words, retrieved with regex
- E — token length of regex

Our final parameters for the environment that were the most successful are as follows

- Index mismatch $a = -100$
- Mismatch in words $b = -10000$
- Penalty for length $c = -1$

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:

$$R = a \cdot (F1 - 1) + b \cdot (A - 1) + c \cdot \sum (A \oplus B) + d \cdot |C - D| + e \cdot E \quad (2)$$

where:

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

Our final parameters for the environment that were the most successful are as follows

- The weight of the F1 metric $a = 100$
- The weight of the Accuracy metric $b = 100$
- Index mismatch $c = 100$
- Mismatch in words $d = -1$
- Penalty for length $e = -0.1$

4 Reinforcement Learning algorithms

We decided to check three RL algorithms: 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:

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

where θ denotes the parameters of policy neural network.

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

$$a_t^* = \arg\max_a \pi_\theta(a|s_t) \quad (4)$$

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

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

$$\begin{aligned} J(\theta) &= \mathbb{E}_{(s_t, a_t) \sim P_\theta(s_t, a_t)} r(s_1 a_1 \dots s_N a_N) = \sum_{s_1 a_1 \dots s_N a_N} P_\theta(s_1 a_1 \dots s_N a_N) R_N \\ &= \sum_{s_1 a_1 \dots s_N a_N} p(s_1) \prod_t \pi_\theta(a_t|s_t) p(s_{t+1}|s_t, a_t) R_N. \end{aligned} \quad (5)$$

Here for the simplicity it is assumed, that N — total length of episode. 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 Equation (5) can be simplified to

$$J(\theta) = \sum_{s_1 a_1 \dots s_N a_N} \prod_t \pi_\theta(a_t|s_t) R_N. \quad (6)$$

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

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

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

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

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 Equations (7) and (8), and is defined as

$$Z(\theta) = \nabla_\theta J(\theta) - \beta \nabla_\theta H(\theta), \quad (9)$$

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.

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

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 Equation (10), the formula for advantage becomes the following:

$$A(s_t, a_t) = R_N - v(s_t). \quad (11)$$

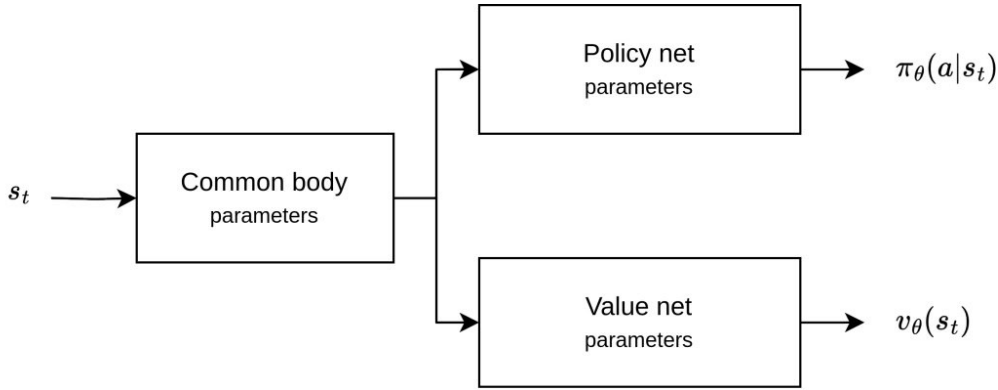


Figure 2: Advantage Actor-Critic (A2C) architecture with shared body.

Policy The policy calculation is similar to REINFORCE case (Equation (3)). The stochastic policy of selection action a_t given state s_t obtained from policy head of neural network depicted on Figure 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 Equation (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 Figure 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 Equation (7):

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

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

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

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

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

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

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

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

where β has the same meaning as in Equation (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 $\mathbb{R}^d \rightarrow (0, 1)$, 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:

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

where θ denotes the parameters of neural network and $\{a_m\}$ — actions space.

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

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:

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

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:

$$Z(\theta) = \nabla_\theta L_Q(\theta). \quad (16)$$

5 Experiments

We have decided to check our models on three different experiments: extraction of single number (low complexity), extraction of word consisting of specified set of symbols (medium complexity) and extraction of email (high complexity). Each experiment has corresponding dataset.

Note that in experiments DQN approach is not presented as it demonstrates pure performance on toy tasks during development. The main reason is that DQN is value-based approach (while REINFORCE and A2C are policy-based) what means the exploration-exploitation trade-off should be explicitly defined. However, we fails to find the good *epsilon*-greedy strategy: either agent explored too much and failed to converge, or agent started exploiting too early and failed to explore all actions. Therefore, we decided to abandon DQN and switch to algorithms that update the policy directly.

5.1 Single number retrieval

The task is to retrieve single number from the string of characters. Number is defined as one or more digit.

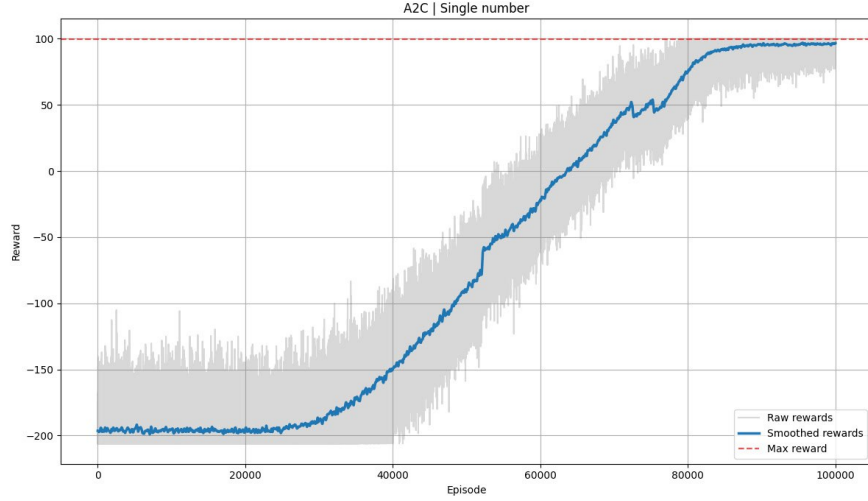


Figure 3: A2C convergence on Single Number Retrieval problem

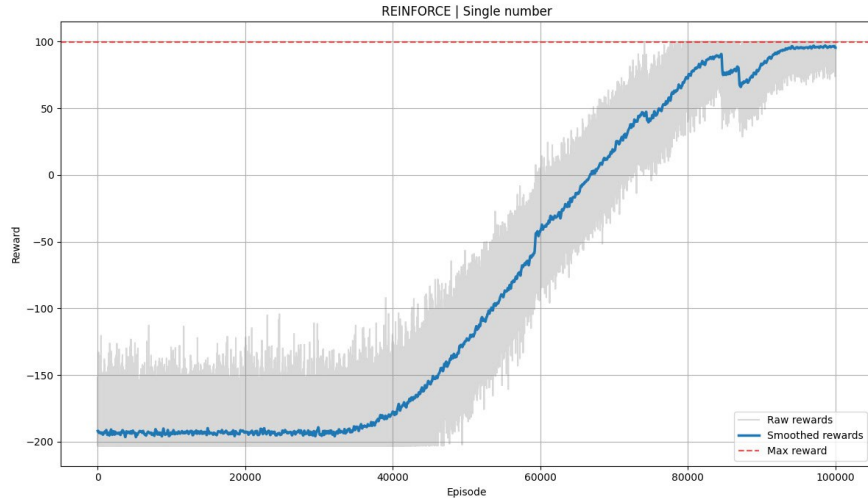


Figure 4: REINFORCE convergence on Single Number Retrieval problem

Agent	Achieved Regex	Total Reward
A2C	<code>\d+</code>	49899.99
REINFORCE	<code>\d+</code>	49899.99
Random agent	<code>v1\+.r</code>	-66791.97

Figure 5: Final results for Single Number Retrieval problem. *Total Reward* is the sum of rewards on all dataset samples

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

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

Results Figures 3 and 4 show that both algorithms converged to the quite high positive rewards. Figure 5 demonstrates that obtained regular expressions are indeed correct.

5.2 Word from subset of symbols retrieval

The task is to retrieve a word which consists of a certain subset of symbols from the string. Word is a string with one or more symbols.

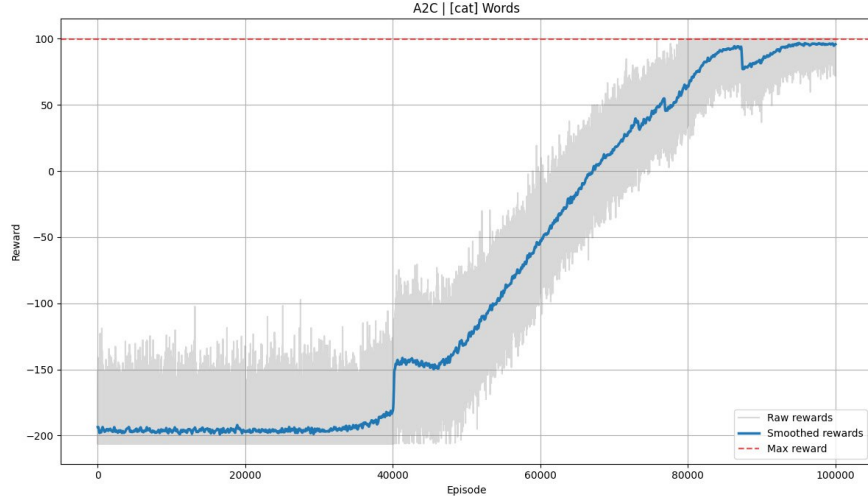


Figure 6: A2C convergence on Word Retrieval problem

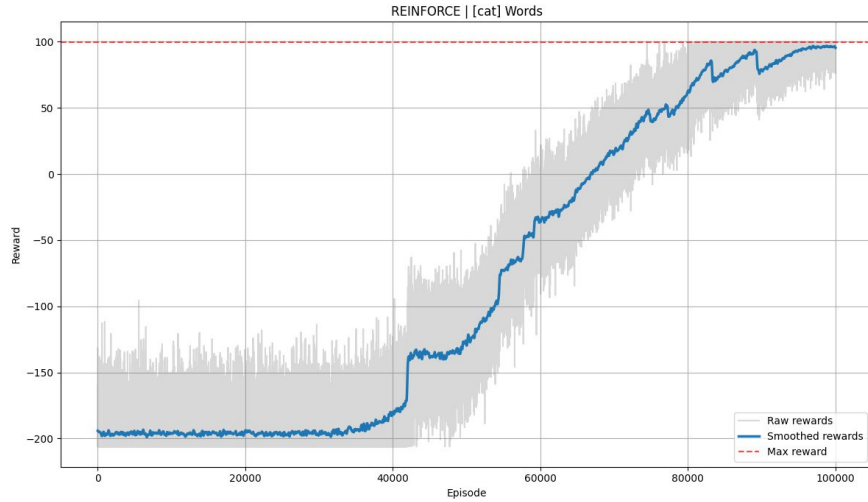


Figure 7: REINFORCE convergence on Word Retrieval problem

Agent	Achieved Regex	Total Reward
A2C	[atc]+	49749.99
REINFORCE	[tca]+	49749.99
Random agent	7CbQr	-65763.80

Figure 8: Final results for Word Retrieval problem. *Total Reward* is the sum of rewards on all dataset samples

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 regex We want our RL agent to simulate the application of `[cat]+` regular expression. Note that actually every combination of letters ‘c’, ‘a’ and ‘t’ can be used inside square brackets.

Results Figures 6 and 7 show that both algorithms converged to the quite high positive rewards. Figure 8 demonstrates that obtained regular expressions are indeed correct.

5.3 Email retrieval

The task is to retrieve an email from the string. The main assumption is email can consist only from english letters (both lower and upper cases).

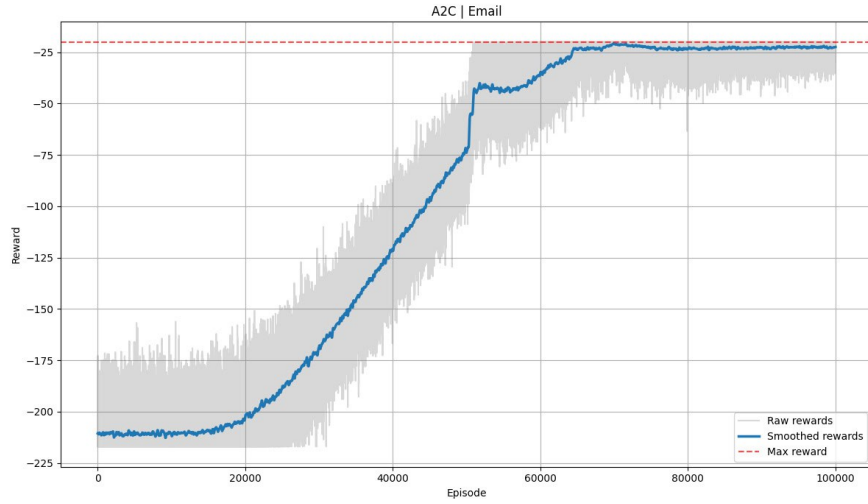


Figure 9: A2C convergence on Word Retrieval problem

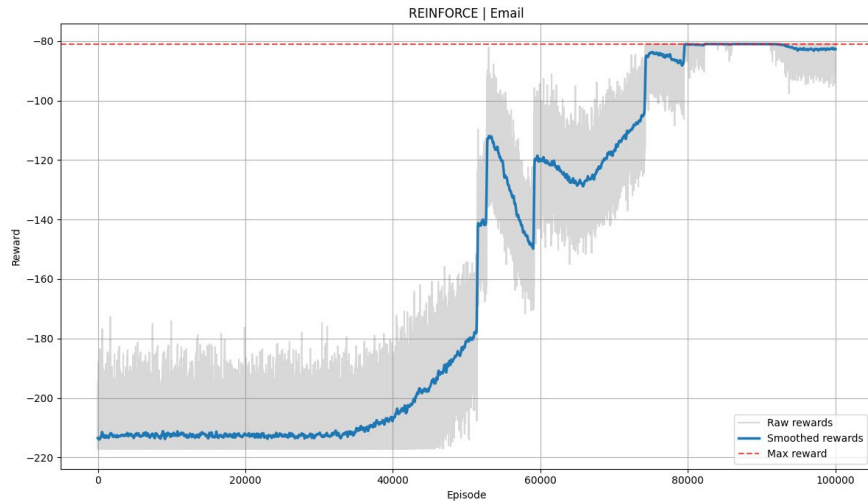


Figure 10: REINFORCE convergence on Word Retrieval problem

Agent	Achieved Regex	Total Reward
A2C	<code>\w+@.*</code>	-9430.51
REINFORCE	<code>@.*</code>	-40971.12
Random agent	<code>Q:Cn0<</code>	-76470.91

Figure 11: Final results for Email Retrieval problem. *Total Reward* is the sum of rewards on all dataset samples

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

Results Figures 9 and 10 show that both algorithms converged to some non-optimal regular expressions. However, Figure 11 demonstrates that obtained regular expressions are more than reasonable though are not fully correct.

6 Results and Discussion

In this project our team successfully formulated regex generation as RL problem, implemented environment and tested several approaches Learning on our dataset, the some agents successfully learned how to get numbers and words consisting of certain letters. However, the agents cannot handle the formation of more complex expressions, for example, email extraction. While some of agents showed not the best performance, we proved, that such task can be indeed be solved via Reinforcement Learning.

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.