Re-Gen: Generating Regex through Modern Architectural Optimizations

Kunal Agarwal
University of California, Berkeley
kagarwal@berkeley.edu

Parth Baokar
University of California, Berkeley
parthbaokar@berkeley.edu

Abstract

This paper explores the latest research in the space of regular expression generation from text descriptions. It also proposes a few enhancements that could be used to improve upon the most current models.

1 Introduction

Regular expressions (Regex) are a fundamental part of the text processing and analysis toolkit. Regex is widely used in preprocessing steps for textual data from removing punctuation to input validation in form fields. Regular expressions offer a powerful syntax to abstract common patterns within text, but its expressiveness comes at the cost of a high learning curve. Small idiosyncrasies when creating Regex make it difficult for beginners to learn to build their own expressions, and can continue to be challenging for the experienced (Friedl, 2002). The consequences for incorrect regex can be drastic as well as failure in input validation can open applications to several security flaws.

However, the mechanics of creating an expression through the combination of metacharacters, grouping characters, and ASCII characters are identical to that of the processes underlying word and sentence formation in natural language. Regex can be considered a language in and of itself, so it should be possible to parse specifications in English and synthesize regular expression that satisfies the constraints of the request in a process not unlike translation between two natural languages. A regex generator from natural language presents itself as a useful tool in education and engineering environments. There is already prior work in this area, but it should be possible to do better with recent advances in natural language processing.

2 Related Work

There are two major approaches that have been taken in translating natural language queries into a corresponding regular expression: a syntactic or semantic approach. The syntactic approach frames this program synthesis question as a machine translation task; it optimizes an objective function that penalizes translations that are not syntactically equivalent (do not have the exact same characters). A semantic approach uses an objective function that encourages translations that have the same inherent meaning.

One of the first truly successful attempts for translating a sentence to an expression was SemUnify (Kushman and Barzilay, 2013), but it relied on prior domain knowledge and so it was limited in its application. The followup to this was a generalized regex synthesis model called Deep-Regex (Locascio et al., 2016) that took advantage of the advent of deep learning. Deep-Regex utilizes a recurrent neural network architecture with attention for the translation task, specifically using LSTM (Long Short-term Memory) cells operating after a word-embedding layer. The LSTMs consist of encoders and decoders which encode the inputted embeddings for the input text description which are decoded into the embeddings for the outputted regular expression. However, since Deep-Regex subscribes to the syntactic approach, the optimal parameters are generated through maximum likelihood estimation (MLE) which discourages the translation of semantically equivalent expressions and leads to unnecessary drops in accuracy.

The current leading language model for regex synthesis, SemRegex (Zhong et al., 2018), relies on a semantics based approach to determine the correctness of a regular expression. They use policy gradient methods (Williams, 1992) to determine the gradient of a semantic based objective (as opposed

to the MLE) and subsequently optimize. Through the use of a reward function r, maximizing the objective function encourages semantic correctness (Zhong et al., 2018). $r(\cdot)$ is defined as 1 if the predicted regular expression is semantically equivalent with the label for the given natural language query (the ground truth), and 0 otherwise. To determine if two regular expressions are semantically equivalent, (Zhong et al., 2018) converts the regular expressions to their minimum DFA (Deterministic Finite Automaton). If these minimum DFAs are equivalent, then the regular expressions themselves are equivalent.

Since there are an infinite number of regular expressions that may have the same minimum DFA as the ground truth, (Zhong et al., 2018) utilizes Monte Carlo sampling to estimate the expected reward of the output of a model. This estimate is then used by the REINFORCE policy gradient model (Williams, 1992) which utilizes a baseline of the mean reward of the samples to reduce variance. The final gradient estimate looks like the following:

$$\nabla_\theta J(\theta) \approx$$

$$\sum_{(S^{(i)},R^{(i)})} \sum_{j=1}^{M} \frac{1}{M}(r(R_j) - b)\nabla_\theta \log p_\theta(R_j|S^{(i)})$$

for $R_1, ..., R_M$ where R_i is a regular expression sampled from the model, and $b = \sum_{j'}^{M} \frac{1}{M} r(R_{j'})$ (Zhong et al., 2018).

More recently, a new semantics-based model SoftRegex (Park et al., 2019) emerged as a an extension of SemRegex. While SemRegex was revolutionary in its approach to Regex generation, it was limited by its computational feasibility. Determining the semantic equivalence of two regular expressions through conversion to DFAs is a known PSPACE-complete problem, and SemRegex's reward function relied on on this step. Furthermore, the reward function was limited in its expressiveness, only outputting a binary value which constrained its ability to describe a spectrum of similarity between two regular expressions. SoftRegex uses an LSTM-based neural network architecture, EQ_Reg, to estimate the similarity between two regular expressions, assigning a probability of similarity between the expressions, the output of which is the value of the reward function. By softening for the comparison of two regular expressions, SoftRegex trades a negligent drop in performance while greatly speeding up training time (Park et al., 2019).

3 Outline for Re-Gen

We hope to improve upon on SoftRegex through architectural changes in computing the similarity score and optimization of the policy gradient add a different optimization to improve accuracy and increase speed.

The EQ_Reg model takes two regular expressions as input and passes it through each of their own LSTMs to generate a latent vector representation. The latent vectors are concatenated and passed through a fully connected layer to generate the similarity measurement. Instead of using an RNN architecture to generate latent vector representations, we can switch to a transformer-based architecture to encode words. Transformer architecture generally outperforms RNNs for longer sequences due to positional embeddings, and have the advantage of pretraining of large corpuses. As additional investigations, we will experiment with adding more downstream fully connected layer and can create an ensemble model for computing the final similarity score.

Now, looking at improving the REINFORCE algorithm, there are a number of different optimizations that could be used. One particular place to improve upon is using a more optimal baseline. While the average baseline used in SemRegex and SoftRegex generally works well (Williams, 1992), we want to explore how much of a performance improvement there would be if the optimal baseline is used. Looking at the gradient estimate defined earlier, we would want to replace the baseline b with this optimal baseline instead of just using the average reward. The optimal baseline, as derived in (Peters and Schaal, 2008), would look like:

$$b = \frac{\sum_{j=1}^{M} \frac{1}{M}(\nabla_\theta \log p_\theta(R_j|S^{(i)}))^2 r(R_j)}{\sum_{j=1}^{M} \frac{1}{M}(\nabla_\theta \log p_\theta(R_j|S^{(i)}))^2}$$

By using the optimal baseline, we are decreasing the variance of the policy gradient model without increasing bias, which should provide improvements in the performance of the model. However, since there is more computation required to compute the baseline for each sampled expression at each iteration, there is far more computation, which may slow down the algorithm.

The authors (Parth and Kunal) are both working on this project. They will divide up the work equally and work on different parts of the project together. This includes writing the code, running experiments, and writing the paper.

References

Jeffrey E.F. Friedl. 2002. *Mastering Regular Expressions*. O'Reilly Media, Inc.".

Nate Kushman and Regina Barzilay. 2013. Using semantic unification to generate regular expressions from natural language. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 826–836, Atlanta, Georgia. Association for Computational Linguistics.

Nicholas Locascio, Karthik Narasimhan, Eduardo DeLeon, Nate Kushman, and Regina Barzilay. 2016. Neural generation of regular expressions from natural language with minimal domain knowledge. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1918–1923, Austin, Texas. Association for Computational Linguistics.

Jun-U Park, Sang-Ki Ko, Marco Cognetta, and Yo-Sub Han. 2019. SoftRegex: Generating regex from natural language descriptions using softened regex equivalence. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 6425–6431, Hong Kong, China. Association for Computational Linguistics.

Jan Peters and Stefan Schaal. 2008. Reinforcement learning of motor skills with policy gradients. *Neural Networks*, 21:682–697.

Ronald J. Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229—-256.

Zexuan Zhong, Jiaqi Guo, Wei Yang, Jian Peng, Tao Xie, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2018. 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, Brussels, Belgium. Association for Computational Linguistics.