Neural Generation of Regular Expressions from Natural Language with Minimal Domain Knowledge

Nicholas Locascio
CSAIL, MIT
njl@mit.edu

Karthik Narasimhan
CSAIL, MIT
karthikn@mit.edu

Eduardo DeLeon
CSAIL, MIT
edeleon4@mit.edu

Nate Kushman
Microsoft
nate@kushman.org

Regina Barzilay
CSAIL, MIT
regina@csail.mit.edu

Abstract

This paper explores the task of translating natural language queries into regular expressions which embody their meaning. In contrast to prior work, the proposed neural model does not utilize domain-specific crafting, learning to translate directly from a parallel corpus. To fully explore the potential of neural models, we propose a methodology for collecting a large corpus[1] of regular expression, natural language pairs. Our resulting model achieves a performance gain of 19.6% over previous state-of-the-art models.

1 Introduction

This paper explores the task of translating natural language text queries into regular expressions which embody their meaning. Regular expressions are built into many application interfaces, yet most users of these applications have difficulty writing them (Friedl, 2002). Thus a system for automatically generating regular expressions from natural language would be useful in many contexts. Furthermore, such technologies can ultimately scale to translate into other formal representations, such as program scripts (Raza et al., 2015).

Prior work has demonstrated the feasibility of this task. Kushman and Barzilay (2013) proposed a model that learns to perform the task from a parallel corpus of regular expressions and the text descriptions. To account for the given representational disparity between formal regular expressions and natural language, their model utilizes a domain specific component which computes the semantic equivalence between two regular expressions. Since their model relies heavily on this component, it cannot be readily applied to other formal representations where such semantic equivalence calculations are not possible.

In this paper, we reexamine the need for such specialized domain knowledge for this task. Given the same parallel corpus used in Kushman and Barzilay (2013), we use an LSTM-based sequence to sequence neural network to perform the mapping. Our model does not utilize semantic equivalence in any form, or make any other special assumptions about the formalism. Despite this and the relatively small size of the original dataset (824 examples), our neural model exhibits a small 0.1% boost in accuracy.

To further explore the power of neural networks, we created a much larger public dataset, **NL-RX**. Since creation of regular expressions requires specialized knowledge, standard crowd-sourcing methods are not applicable here. Instead, we employ a two-step generate-and-paraphrase procedure that circumvents this problem. During the generate step, we use a small but expansive manually-crafted grammar that translates regular expression into natural language. In the paraphrase step, we rely on crowd-sourcing to paraphrase these rigid descriptions into more natural and fluid descriptions. Using this methodology, we have constructed a corpus of 10,000 regular expressions, with corresponding verbalizations.

Our results demonstrate that our sequence to sequence model significantly outperforms the domain specific technique on the larger dataset, reaching a

[1] The corpus and code used in this paper is available at https://github.com/nicholaslocascio/deep-regex

Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, pages 1918–1923,
Austin, Texas, November 1-5, 2016.

gain of 19.6% over of the state-of-the-art technique.

2 Related Work

Regular Expressions from Natural Language There have been several attempts at generating regular expressions from textual descriptions. Early research into this task used rule-based techniques to create a natural language interface to regular expression writing (Ranta, 1998). Our work, however, is closest to Kushman and Barzilay (2013). They learned a semantic parsing translation model from a parallel dataset of natural language and regular expressions. Their model used a regular expression-specific semantic unification technique to disambiguate the meaning of the natural language descriptions. Our method is similar in that we require only description and regex pairs to learn.r However, we treat the problem as a direct translation task without applying any domain-specific knowledge.

Neural Machine Translation Recent advances in neural machine translation (NMT) (Bahdanau et al., 2014; Devlin et al., 2014) using the framework of sequence to sequence learning (Sutskever et al., 2014) have demonstrated the effectiveness of deep learning models at capturing and translating language semantics. In particular, recurrent neural networks augmented with attention mechanisms (Luong et al., 2015) have proved to be successful at handling very long sequences. In light of these successes, we chose to model regular expression generation as a neural translation problem.

3 Regex Generation as Translation

We use a Recurrent Neural Network (RNN) with attention (Mnih et al., 2014) for both encoding and decoding (Figure 1).

Let $W = w_1, w_2 ... w_m$ be the input text description where each w_i is a word in the vocabulary. We wish to generate the regex $R = r_1, r_2, ... r_n$ where each r_i is a character in the regex.

We use Long Short-Term Memory (LSTM) (Hochreiter and Schmidhuber, 1997) cells in our model, the transition equations for which can be summarized as:

$$\begin{aligned} i_t &= \sigma(U^{(i)} x_t + V^{(i)} h_{t-1} + b^{(i)}), \\ f_t &= \sigma(U^{(f)} x_t + V^{(f)} h_{t-1} + b^{(f)}), \\ o_t &= \sigma(U^{(o)} x_t + V^{(o)} h_{t-1} + b^{(o)}) \\ z_t &= \tanh(U^{(z)} x_t + V^{(z)} h_{t-1} + b^{(z)}) \\ c_t &= i_t \odot z_t + f_t \odot c_{t-1} \\ h_t &= o_t \odot \tanh(c_t) \end{aligned} \tag{1}$$

where σ represents the sigmoid function and $\odot$ is elementwise multiplication. i_t refers to the input gate, f_t is the forget gate, and o_t is the output gate at each time step. The U and V variables are weight matrices for each gate while the b variables are the bias parameters. The input x_t is a word (w_t) for the encoder and the previously generated character r_{t-1} for the decoder.

The attention mechanism is essentially a 'soft' weighting over the encoder's hidden states during decoding:

$$\alpha_t(e) = \frac{\exp(\text{score}(h_t, h_e))}{\sum_{e'} \exp(\text{score}(h_t, h_{e'}))}$$

where h_e is a hidden state in the encoder and *score* is the scoring function. We use the general attention matrix weight (as described in (Luong et al., 2015)) for our scoring function. The outputs of the decoder r_t are generated using a final softmax layer.

Our model is six layers deep, with one word embedding layer, two encoder layers, two decoder layers, and one dense output layer. Our encoder and decoder layers use a stacked LSTM architecture with a width of 512 nodes. We use a global attention mechanism (Bahdanau et al., 2014), which considers all hidden states of the encoder when computing the model's context vector. We perform standard dropout during training (Srivastava et al., 2014) after every LSTM layer with dropout probability equal to 0.25. We train for 20 epochs, utilizing a minibatch size of 32, and a learning-rate of 1.0. The learning rate is decayed by a factor 0.5 if evaluation perplexity does not increase.

4 Creating a Large Corpus of Natural Language / Regular Expression Pairs

Previous work in regular expression generation has used fairly small datasets for training and evaluation.

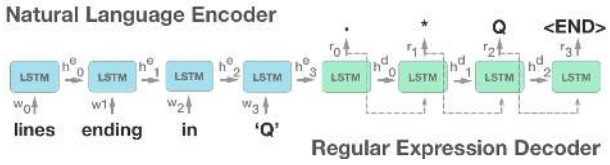

Figure 1: Deep-Regex Encoder-Decoder setup. Blue cells represent the encoder and the green ones represent the decoder.

Non-Terminals		
x & y → x and y	x | y → x or y	∼(x) → not x
.*x.*y → x followed by y	.*x.* → contains x	x{N,} → x, N or more times
x& y& z → x and y and z	x | y | z → x or y or z	x{1,N} → x, at most N times
x.* → starts with x	.*x → ends with x	\b x\b → words with x
(x)+ → x, at least once	(x)* → x, zero or more times	x → only x

Terminals		
[AEIOUaeiou] → a vowel	[0-9] → a number	word → the string 'word'
[A-Z] → an uppercase letter	[a-z] → a lowercase letter	. → a character

Table 1: Regex → Synthetic Grammar for Data Generation

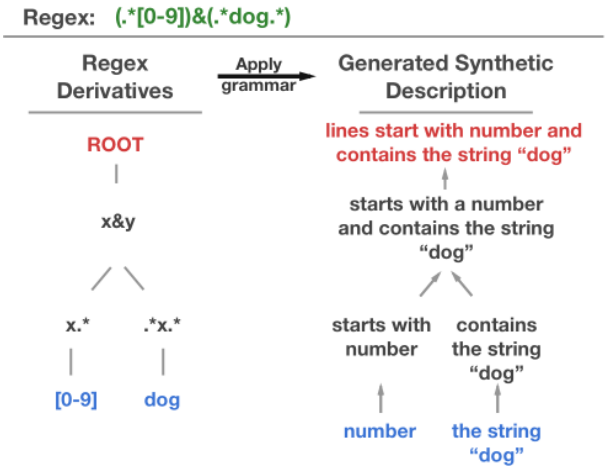

Figure 2: Process for generating Synthetic Descriptions from Regular Expressions. Grammar rules from Table 1 are applied to a node's children and the resulting string is passed to the node's parent.

Synthetic:	lines not words with starting with a capital letter
Paraphrased:	lines that do not contain words that begin with a capital letter
Regex:	∼ (\b([A-Z])(.*)\b)

Table 2: NL-RX Text Descriptions and Regular Expression

In order to fully utilize the power of neural translation models, we create a new large corpus of regular expression, natural language pairs titled **NL-RX**.

The challenge in collecting such corpora is that typical crowdsourcing workers do not possess the specialized knowledge to write regular expressions. To solve this, we employ a two-step generate-and-paraphrase procedure to gather our data. This technique is similar to the methods used by Wang et al. (2015) to create a semantic parsing corpus.

In the generate step, we generate regular expression representations from a small manually-crafted grammar (Table 1). Our grammar includes 15 non-terminal derivations and 6 terminals and of both basic and high-level operations. We identify these via frequency analysis of smaller datasets from previous work (Kushman and Barzilay, 2013). Every grammar rule has associated verbalizations for both regular expressions and language descriptions. We use this grammar to stochastically generate regular expressions and their corresponding synthetic language descriptions. This generation process is shown in Figure 2.

While the automatically generated descriptions are semantically correct, they do not exhibit richness and variability of human-generated descriptions. To obtain natural language (non-synthetic) descriptions, we perform the paraphrase step. In this step, Mechanical Turk (Amazon, 2003) human workers paraphrase the generated synthetic descriptions into the fluent verbalizations.

NL-RX Using the procedure described above, we create a new public dataset (NL-RX) comprising of 10,000 regular expressions and their corresponding natural language descriptions. Table 2 shows an example from our dataset.

Our data collection procedure enables us to create a substantially larger and more varied dataset than previously possible. Employing standard crowdsource workers to paraphrase is more cost-efficient and scalable than employing professional regex programmers, enabling us to create a much larger dataset. Furthermore, our stochastic generation of regular expressions from a grammar results in a more varied dataset because it is not subject to the bias of human workers who, in previous work, wrote many duplicate examples (see Results).

Corpora Statistics Our seed regular expression grammar (Table 1), covers approximately 85% of the original KB13 regular expressions. Additionally, NL-RX contains exact matches with 30.1% of the original KB13 dataset regular expressions. This means that 248 of the 824 regular expressions in the

Verbalization	Frequency
'the word x'	12.6%
'x before y'	9.1%
'x or y'	7.7%
'x, at least once'	6.2%
'a vowel'	5.3%

Table 3: Top Frequent Verbalizations from NL-RX

KB13 dataset were also in our dataset. The average length of regular expressions in NL-RX is 25.9 characters, the average in the KB13 dataset is 19.7 characters. We also computed the grammar breakdown of our NL-RX. The top 5 occurring terminals in our generated regular expressions are those corresponding with the verbalizations shown in Table 3.

Crowdsourcing details We utilize Mechanical Turk for our crowdsource workers. A total of 243 workers completed the 10,000 tasks, with an average task completion time of 101 seconds. The workers proved capable of handling complex and awkward phrasings, such as the example in Table 2, which is one of the most difficult in the set.

We applied several quality assurance measures on the crowdsourced data. Firstly, we ensured that our workers performing the task were of high quality, requiring a record of 97% accuracy over at least 1000 other previous tasks completed on Mechanical Turk. In addition, we ran automatic scripts that filtered out bad submissions (e.g. submissions shorter than 5 characters). In all, we rejected 1.1% of submissions, which were resubmitted for another worker to complete. The combination of these measures ensured a high quality dataset, and we confirmed this by performing a manual check of 100 random examples. This manual check determined that approximately 89% of submissions were a correct interpretation, and 97% were written in fluent English.

5 Experiments

Datasets We split the 10,000 regexp and description pairs in NL-RX into 65% train, 10% dev, and 25% test sets.

In addition, we also evaluate our model on the dataset used by Kushman and Barzilay (2013) (KB13), although it contains far fewer data points (824). We use the 75/25 train/test split used in their work in order directly compare our performance to theirs.

Training We perform a hyper-parameter grid-search (on the dev set), to determine our model hyper-parameters: learning-rate = 1.0, encoder-depth = 2, decoder-depth = 2, batch size = 32, dropout = 0.25. We use a Torch (Collobert et al., 2002) implementation of attention sequence to sequence networks from (Kim, 2016). We train our models on the train set for 20 epochs, and choose the model with the best average loss on the dev set.

Evaluation Metric To accurately evaluate our model, we perform a *functional* equality check called DFA-Equal. We employ functional equality because there are many ways to write equivalent regular expressions. For example, (a|b) is functionally equivalent to (b|a), despite their string representations differing. We report DFA-Equal accuracy as our model's evaluation metric, using Kushman and Barzilay (2013)'s implementation to directly compare our results.

Baselines We compare our model against two baselines:

BoW-NN: BoW-NN is a simple baseline that is a Nearest Neighbor classifier using Bag Of Words representation for each natural language description. For a given test example, it finds the closest cosine-similar neighbor from the training set and uses the regexp from that example for its prediction.

Semantic-Unify: Our second baseline, Semantic-Unify, is the previous state-of-the-art model from (Kushman and Barzilay, 2013), explained above. [2]

6 Results

Our model significantly outperforms the baselines on the NL-RX dataset and achieves comparable performance to *Semantic Unify* on the KB13 dataset (Table 4). Despite the small size of KB13, our model achieves state-of-the-art results on this very resource-constrained dataset (824 examples). Using NL-RX, we investigate the impact of training data size on our model's accuracy. Figure 3 shows how

[2]We trained and evaluated Semantic-Unify in consultation with the original authors.

Models	NL-RX-Synth		NL-RX-Turk		KB13
	Dev	Test	Dev	Test	Test
BoW NN	31.7%	30.6%	18.2%	16.4%	48.5%
Semantic-Unify	41.2%	46.3%	39.5%	38.6%	65.5%
Deep-RegEx	**85.75%**	**88.7%**	**61.2%**	**58.2%**	**65.6%**

Table 4: DFA-Equal accuracy on different datasets. **KB13**: Dataset from Kushman and Barzilay(2013), **NL-RX-Synth**: NL Dataset with original synthetic descriptions, **NL-RX-Turk**: NL Dataset with Mechanical Turk paraphrased descriptions. Best scores are in bold.

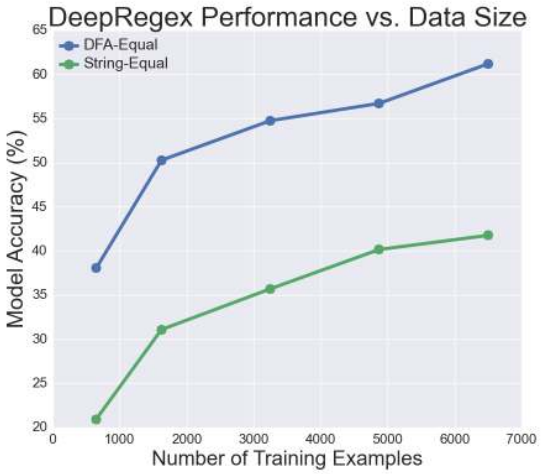

Figure 3: Our model's performance, like many deep learning models, increases significantly with larger datasets. **String-Equal**:Accuracy on direct string match, **DFA-Equal**:Accuracy using the DFA-Equal evaluation.

our model's performance improves as the number of training examples grows.

Differences in Datasets Keeping the previous section in mind, a seemingly unusual finding is that the model's accuracy is higher for the smaller dataset, KB13, than for the larger dataset, NL-RX-Turk. On further analysis, we learned that the KB13 dataset is a much less varied and complex dataset than NL-RX-Turk. KB13 contains many duplicates, with only 45% of its regular expressions being unique. This makes the translation task easier because over half of the correct test predictions will be exact repetitions from the training set. In contrast, NL-RX-Turk does not suffer from this variance problem and contains 97% unique regular expressions. The relative easiness of the KB13 dataset is further illustrated by the high performance of the Nearest-Neighbor baselines on the KB13 dataset.

7 Conclusions

In this paper we demonstrate that generic neural architectures for generating regular expressions outperform customized, heavily engineered models. The results suggest that this technique can be employed to tackle more challenging problems in broader families of formal languages, such as mapping between language description and program scripts. We also have created a large parallel corpus of regular expressions and natural language queries using typical crowd-sourcing workers, which we make available publicly.

Acknowledgments

We thank the anonymous reviewers for their helpful feedback and suggestions.

References

[Amazon2003] Amazon. 2003. Mechanical turk. https://mturk.com.

[Bahdanau et al.2014] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473.

[Collobert et al.2002] Ronan Collobert, Samy Bengio, and Johnny Marithoz. 2002. Torch: A modular machine learning software library. https://torch.ch.

[Devlin et al.2014] Jacob Devlin, Rabih Zbib, Zhongqiang Huang, Thomas Lamar, Richard M Schwartz, and John Makhoul. 2014. Fast and robust neural network joint models for statistical machine translation. In *ACL (1)*, pages 1370–1380. Citeseer.

[Friedl2002] Jeffrey EF Friedl. 2002. *Mastering regular expressions*. " O'Reilly Media, Inc.".

[Hochreiter and Schmidhuber1997] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

[Kim2016] Yoon Kim. 2016. Seq2seq-attn. https://github.com/harvardnlp/seq2seq-attn.

[Kushman and Barzilay2013] Nate Kushman and Regina Barzilay. 2013. Using semantic unification to generate regular expressions from natural language. North American Chapter of the Association for Computational Linguistics (NAACL).

[Luong et al.2015] Thang Luong, Hieu Pham, and Christopher D. Manning. 2015. Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages

1412–1421, Lisbon, Portugal, September. Association for Computational Linguistics.

[Mnih et al.2014] Volodymyr Mnih, Nicolas Heess, Alex Graves, et al. 2014. Recurrent models of visual attention. In *Advances in Neural Information Processing Systems*, pages 2204–2212.

[Ranta1998] Aarne Ranta. 1998. A multilingual natural-language interface to regular expressions. In *Proceedings of the International Workshop on Finite State Methods in Natural Language Processing*, pages 79–90. Association for Computational Linguistics.

[Raza et al.2015] Mohammad Raza, Sumit Gulwani, and Natasa Milic-Frayling. 2015. Compositional program synthesis from natural language and examples. *International Joint Conference on Artificial Intelligence (IJCAI)*.

[Srivastava et al.2014] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

[Sutskever et al.2014] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.

[Wang et al.2015] Yushi Wang, Jonathan Berant, and Percy Liang. 2015. Building a semantic parser overnight. *Association for Computational Linguistics (ACL)*.