Regex-RNN: Generating Regular Expressions from Natural Language Prompts

Nicholas Locascio, Eduardo de Leon

Problem Setup

Dataset of 824 Regular Expression + Natural Language Pairs

Natural Language:
lines with the word 'Triple' followed by words that start with 'X'

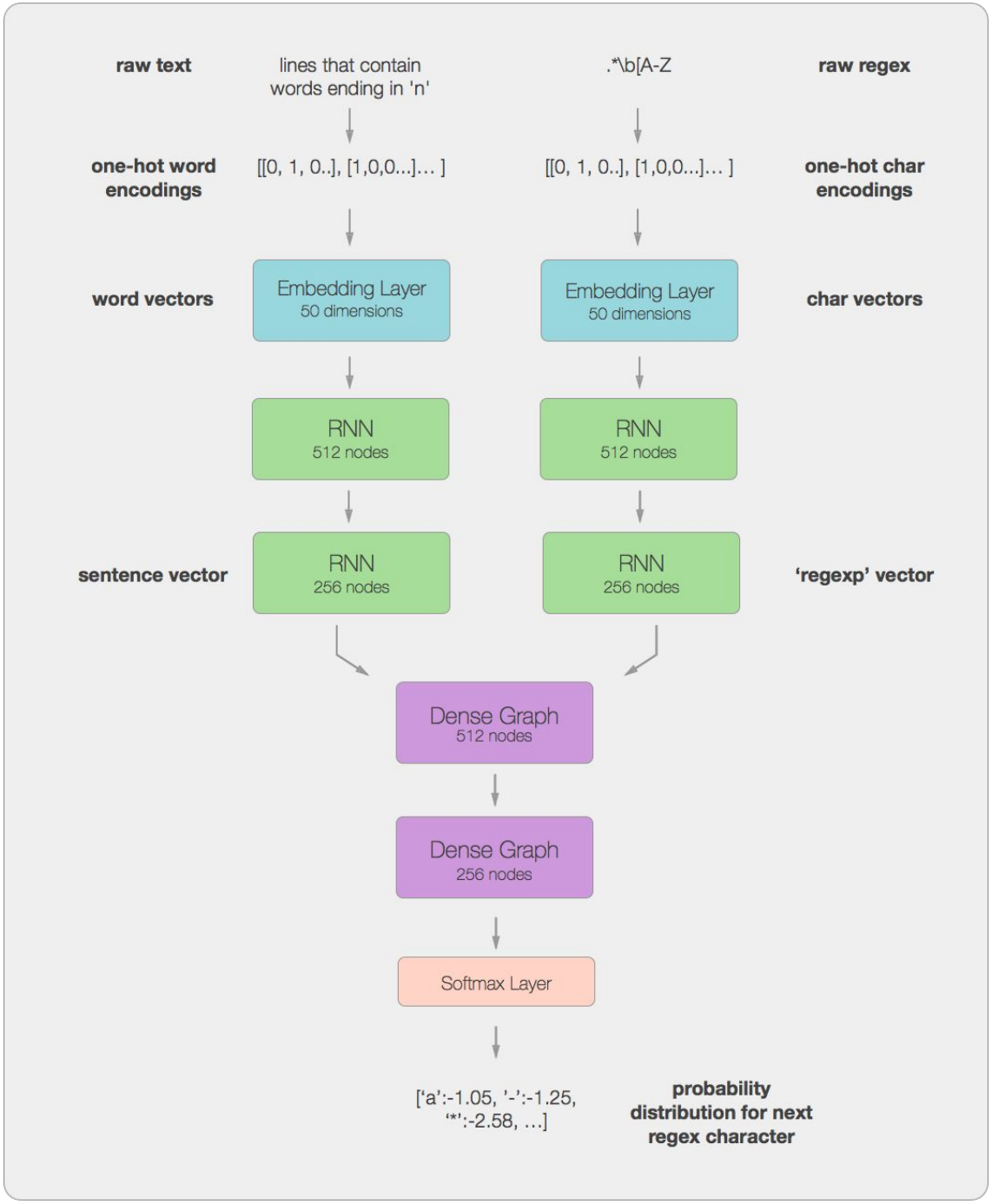
Regex:
.*\bTriple\b.*\bX[A-Za-z]*\b.*

Approach:
From the natural language description, predict the corresponding regex 1 character at a time.

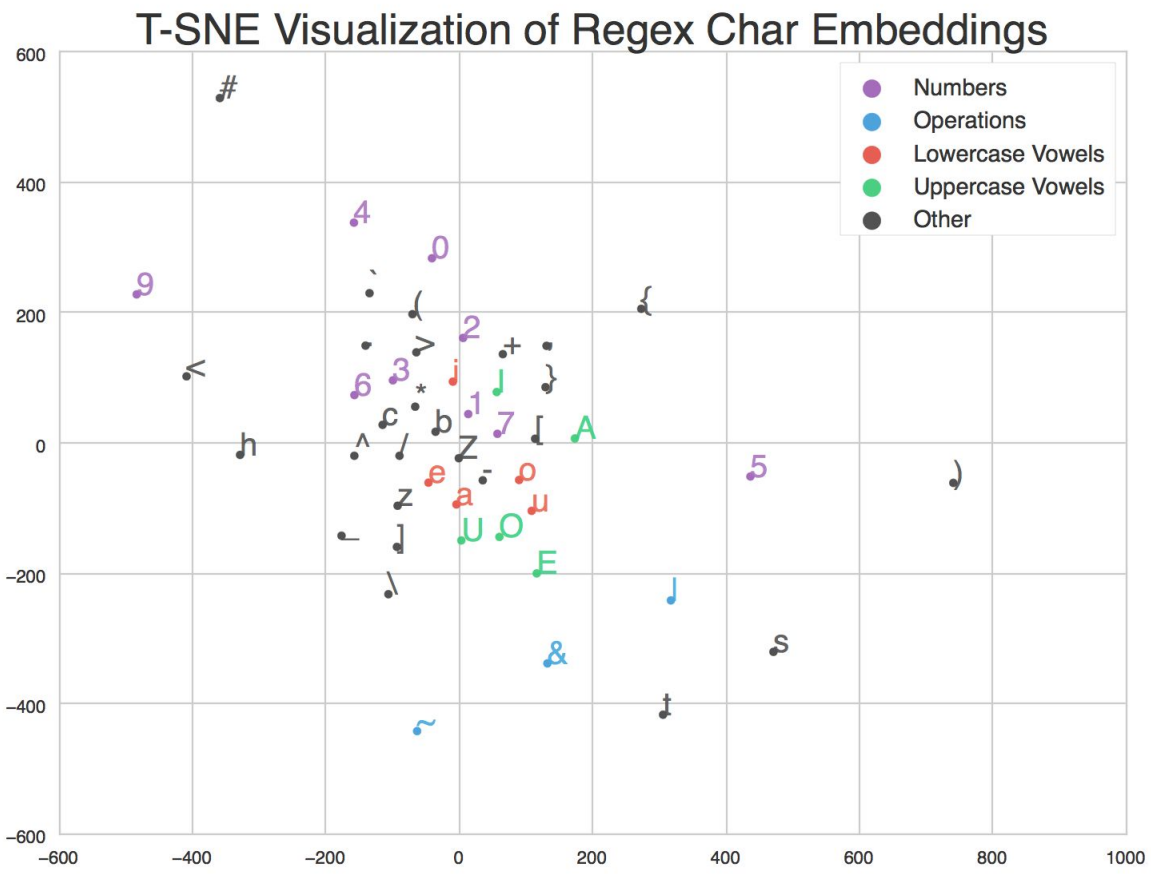
Build character generation model for:
P(next_regex_char | prompt, regex_so_far)

Example:
P(“Z” | “lines with the word triple followed by words that start with ‘x’”, “.*\bTriple\b.*\bX[A-”)

Regex-RNN Model Architecture



Learning Regex Char-Vectors with Word2Vec



Analogies Learned

E - e + u = U
(*capitalization*)
2 - 1 + 3 = 4
(*number order*)
4 - 1 + 3 = 6
(*number magnitudes*)

Generating Complete Regexes: RNN + Viterbi Beam Search

- Our RNN model predicts **next character** probabilities.

- Need to find best **complete path: 'START'->'['->'A'->'-'->'Z'->']'->''END"**

- Path Probability = Π(P(char | prompt , so_far))
= Σ log(sum(P(char | prompt, so_far)))

- Can take greedy path **(BAD Results).** Can take optimal path **(NP Runtime).**

- We use BFS **beam search** to compute a good solution in reasonable time. (Used K=80, keeping track of top 80 paths).

- Since we are considering **variable-length** sequences, we have to normalize our path probability by the path length to get **average log probability per character.** Otherwise, short sequences would always be preferred.

Example Predictions

Natural Language Prompt	**Prediction**	**Answer**	**Key Point**	
lines that contain only three words	(([^A-Za-z])*\b[A-Za-z]+\b([^A-Za-z])*){3}	(([^A-Za-z])*\b[A-Za-z]+\b([^A-Za-z])*){3}	Can work on long strings.	✓
lines using 'lugg' before 'age'	.*lugg.*age.*	.*lugg.*age.*	Can do “entity” replacement.	✓
lines that contain a '?' or an '!'	.*(?|!).*	(.*?.*)|(.*!.*)	Can generate shorter DFA equivalent answers.	✓

Example Mistakes

Natural Language Prompt	Prediction	Answer	Key Point	
lines that have no letters	~(.*[AEIOUaeiou].*)	~(.*[A-Za-z].*)	Can get lost in thought w/ common sequences	X
lines where there are three characters between instances of "ABC" and "WEX"	.*ABC.*.{2}.*WEX.*|.*WEX.*.{3}.*ABC.*	.*ABC.*.{3}.*WEX.*|.*WEX.*.{3}.*ABC.*	Can mix-up numbers.	X
lines having words with 'ro'.	.*((\b[A-Za-z]+\b)&(.*ro)).*	.*\b[A-Za-z]*ro[A-Za-z]*\b.*	Ambiguous Prompt!	X

Results

	Semantic Unification (Kushman, 2013)	UBL Model (Kwiatkowski, 2010)	Regex-RNN Top Result	Regex-RNN Top 5 Results*
DFA Equal	65.5%	36.5%	56.6%	66.5%
Example Equal	x	x	60.6%	70.0%

DFA equal means a regex was only correct if its FSM was exactly equivalent to the answer regex.
Example equal means a regex was only counted correct if it matched all 10 positive and negative examples perfectly.

Future Work

- **Use larger dataset.** Neural nets are data-hungry models and 824 examples is extremely small.
- Use **Sequence-To-Sequence RNN**.
- Keep **shared representations** for some words and characters.
- Use **Attention-based NN**. Our problem is extremely sensitive to context, so attention-based models would help
- Use EM algorithm to compute **word-alignments**.
- Pre-Process regexes for best DFA alignment.

References

"Using Semantic Unification to Generate Regular Expressions from Natural Language", Nate Kushman.
Tom Kwiatkowski, Luke Zettlemoyer, Sharon Goldwater, and Mark Steedman. 2010. Inducing probabilistic ccg grammars from logical form with higher-order unification. In Proceedings of EMNLP.