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

Written by Nicholas Locascio, Karthik Narasimhan and Eduardo DeLeon for EMNLP 2016

Presented by Xia Cui for NLP@UoL
Overview

• Task
  • Translate Natural Language (NL) queries into Regular Expressions (RX) which embody their meaning

• NL-RX Example:
  • NL: lines not words with starting with a capital letter
  • RX: \~(\b([A-Z])(.*)\b)

• Main Difference against the prior work
  • Do not utilize domain-specific crafting, direct translate
Regular Expressions from Natural Language

• Ranta (1998)
  • Rule-based techniques to create NL interface to RX writing

• Kushman and Barzilay (2013)
  • Learn a semantic parsing translation model from NL and RX parallel dataset
  • Use a RE-specific semantic unification techniques to disambiguate the meaning of NL descriptions
  • Similar to proposed approach: only need description and regex pairs to learn
Neural Machine Translation

• Sutskever et al. (2014)
  • Use the framework of sequence to sequence learning

• Luong et al. (2015)
  • Recurrent neural networks augmented with attention mechanisms
  • Good for handling long sequences
Regex Generation as Translation

• Model: 6 layers deep
  • One word embedding layer
  • Two encoder layers
  • Two decoder layers
  • One dense output layer

• Settings
  • Input Text: $W = w_1, w_2... w_m$
  • Output Regex: $R = r_1, r_2, ... r_n$

• Translation Equations:
  • Use Long Short-Term Memory (Hochreiter and Schmidhuber, 1997)

$$i_t = \sigma(U^{(i)} x_t + V^{(i)} h_{t-1} + b^{(i)}),$$
$$f_t = \sigma(U^{(f)} x_t + V^{(f)} h_{t-1} + b^{(f)}),$$
$$o_t = \sigma(U^{(o)} x_t + V^{(o)} h_{t-1} + b^{(o)}),$$
$$z_t = \tanh(U^{(z)} x_t + V^{(z)} h_{t-1} + b^{(z)}),$$
$$c_t = i_t \odot z_t + f_t \odot c_{t-1},$$
$$h_t = o_t \odot \tanh(c_t)$$
Regex Generation as Translation

- Model: 6 layers deep
  - One word embedding layer
  - Two encoder layers
  - Two decoder layers
  - One dense output layer

- Settings
  - Input Text: $W = w_1, w_2 ... w_m$
  - Output Regex: $R = r_1, r_2, ... r_n$

- Translation Equations:
  - Use Long Short-Term Memory (Hochreiter and Schmidhuber, 1997)

$$
i_t = \sigma(U^{(i)} x_t + V^{(i)} h_{t-1} + b^{(i)}),$$
$$f_t = \sigma(U^{(f)} x_t + V^{(f)} h_{t-1} + b^{(f)}),$$
$$o_t = \sigma(U^{(o)} x_t + V^{(o)} h_{t-1} + b^{(o)}),$$
$$z_t = \tanh(U^{(z)} x_t + V^{(z)} h_{t-1} + b^{(z)}),$$
$$c_t = i_t \odot z_t + f_t \odot c_{t-1},$$
$$h_t = o_t \odot \tanh(c_t),$$
$$r_{t-1} = r_t \odot \text{weight matrix},$$
$$b_{t-1} = b_t \odot \text{bias parameter}.$$
Attention Mechanism & Training

• “soft” weighting over the encoder’s hidden states during decoding

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

(Bahdanau et al., 2014)

• \( \mathbf{r}_t \) : outputs from the decoder generated using a final softmax layer

• perform standard dropout during training (Srivastava et al., 2014)
  • dropout probability = 0.25 (after every LSTM layer)
  • train for 20 epochs, minibatch = 32, learning-rate = 1.0
Deep-Regex Encoder-Decoder

Natural Language Encoder

Regular Expression Decoder
Creating a Large Corpus of NL/RX Pairs

• Previous work: fairly small datasets for training and evaluation

• Create a new large corpus of regular expression, natural language pairs titled **NL-RX**
  • Challenge: typical crowdsourcing workers do not possess the specialized knowledge to write regular expressions
  • Solution: two-step generate-and-paraphrase procedure
    • Similar to Wang et al. (2015) for create a semantic parsing corpus
Generate-and-Paraphrase

1. Generate
   • Generate RX representations from a small manual-crafted grammar

<table>
<thead>
<tr>
<th>Non-Terminals</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x &amp; y → x and y</td>
<td>x</td>
</tr>
<tr>
<td>.&quot;x.&quot;y → x followed by y</td>
<td>.&quot;x.&quot; → contains x</td>
</tr>
<tr>
<td>x&amp; y&amp; z → x and y and z</td>
<td>x</td>
</tr>
<tr>
<td>x.&quot; → starts with x</td>
<td>.&quot;x&quot; → ends with x</td>
</tr>
<tr>
<td>(x)+ → x, at least once</td>
<td>(x)* → x, zero or more times</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Terminals</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>[AEIOUaciou] → a vowel</td>
<td>[0-9] → a number</td>
</tr>
<tr>
<td>[A-Z] → an uppercase letter</td>
<td>[a-z] → a lowercase letter</td>
</tr>
</tbody>
</table>
Generate-and-Paraphrase

1. **Generate**
   
   - use this grammar to generate regular expressions and their corresponding synthetic language descriptions
Generate-and-Paraphrase

2. Paraphrase

• Human workers paraphrase generated synthetic descriptions into fluent verbalizations (Obtain Natural Language)

| 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) |

An example in NL-RX datasets
Datasets (NL-RX)

- [https://github.com/nicholaslocascio/deep-regex](https://github.com/nicholaslocascio/deep-regex)
- KB13 (Kushman and Barzilay, 2013)
- NL-RX (with step1)
- NL-RX (with step1&2)

- 10,000 pairs in NL-RX, 824 pairs in KB13
Training

• NL-RX: 65% train, 10% dev and 25% test
• 20 epochs, learning-rate = 1.0, encoder-depth = 2, decoder-depth = 2, batch size = 32, dropout = 0.25
• Implementation of attention sequence to sequence networks (Kim, 2016)
  • https://github.com/harvardnlp/seq2seq-attn

Evaluation

• DFA-Equal: functional equality check
• Baselines
  • BoW-NN: a Nearest Neighbor classifier using Bag Of Words
  • Semantic-Unify: Kushman and Barzilay (2013)
## Results

<table>
<thead>
<tr>
<th>Models</th>
<th>NL-RX-Synth</th>
<th></th>
<th>NL-RX-Turk</th>
<th></th>
<th>KB13</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Dev</td>
<td>Test</td>
<td>Dev</td>
<td>Test</td>
<td>Test</td>
</tr>
<tr>
<td>BoW NN</td>
<td>31.7%</td>
<td>30.6%</td>
<td>18.2%</td>
<td>16.4%</td>
<td>48.5%</td>
</tr>
<tr>
<td>Semantic-Unify</td>
<td>41.2%</td>
<td>46.3%</td>
<td>39.5%</td>
<td>38.6%</td>
<td>65.5%</td>
</tr>
<tr>
<td>Deep-RegEx</td>
<td><strong>85.75%</strong></td>
<td><strong>88.7%</strong></td>
<td><strong>61.2%</strong></td>
<td><strong>58.2%</strong></td>
<td><strong>65.6%</strong></td>
</tr>
</tbody>
</table>

### DeepRegex Performance vs. Data Size

![Graph showing DeepRegex performance vs. data size]