Program Synthesis for Character Level Language Modelling

Pavol Bielik    Veselin Raychev    Martin Vechev

Department of Computer Science, ETH Zurich, Switzerland

CSC2547, Winter 2018
Motivation

- Neural networks are not as effective on structured tasks (e.g., program synthesis).
- Neural network weights are difficult to interpret.
- It is difficult to define sub-models for different circumstances.
TChar is a domain-specific language (DSL) for writing programs that define probabilistic n-gram models and variants. Variants include models trained on subsets of data, queried only when certain conditions are met, used to make certain classes of predictions, etc. Submodels can be composed into a larger model using if-then statements.
Let $f$ be a function (program) from $\text{TChar}$ that takes a prediction position $t$ in a text $x$ and returns a context to predict with. Say

$$x = \text{Dogs are th}_t$$

For example, say $f(t, x) = x_s$ if $x_{t-1}$ is whitespace else $x_{t-2}x_{t-1}$, where $x_s$ is the first character of the previous word.

Then predict $x_t$ using distribution $P(x_t|f(t, x))$. 

Example

Let \( f \) be a function (program) from TChar that takes a prediction position \( t \) in a text \( x \) and returns a context to predict with. Say

\[
x = \text{Dogs are th}_t
\]

For example, say \( f(t, x) = x_s \) if \( x_{t-1} \) is whitespace else \( x_{t-2}x_{t-1} \), where \( x_s \) is the first character of the previous word.

Then predict \( x_t \) using distribution \( P(x_t | f(t, x)) \).

This is just a trigram language model with special behavior for starting characters!
**SimpleProgram:** Use Move and Write instructions to condition the prediction (1), update the program state (2), or determine which branch to choose (3). (e.g., LEFT WRITE_CHAR LEFT WRITE_CHAR provides context for trigram language model).

**SwitchProgram:** Use switch statements to conditionally select appropriate subprograms (e.g., use switch LEFT_WRITE_CHAR to separately handle newline, tabs, special characters, and upper-case characters.)

**StateProgram:** Update the current state and determine which program to execute next based on current state (e.g., use LEFT WRITE_CHAR LEFT WRITE_CHAR that updates state on */ to handle comments separately).
Building Blocks

- **SimpleProgram**: Use Move and Write instructions to condition the prediction (1), update the program state (2), or determine which branch to choose (3). (e.g., LEFT WRITE_CHAR LEFT WRITE_CHAR provides context for trigram language model).

- **SwitchProgram**: Use switch statements to conditionally select appropriate subprograms (e.g., use `switch` LEFT WRITE_CHAR to separately handle newline, tabs, special characters, and upper-case characters.)
Building Blocks

- **SimpleProgram**: Use `Move` and `Write` instructions to condition the prediction (1), update the program state (2), or determine which branch to choose (3). (e.g., `LEFT WRITE_CHAR LEFT WRITE_CHAR` provides context for trigram language model).

- **SwitchProgram**: Use `switch` statements to conditionally select appropriate subprograms (e.g., use `switch` `LEFT WRITE_CHAR` to separately handle newline, tabs, special characters, and upper-case characters.)

- **StateProgram**: Update the current state and determine which program to execute next based on current state (e.g., use `LEFT WRITE_CHAR LEFT WRITE_CHAR` that updates state on */ to handle comments separately).
Given a validation set $D$ and regularization penalty $\Omega$, the learning process is to find a program $p^* \in \text{TChar}$:

$$p^* = \arg \min_p \left[ -\log P(p|D) + \lambda \cdot \Omega(p) \right]$$

- $\text{TChar}$ consists of branches and SimplePrograms.
- Branches are synthesized using the ID3+ algorithm.
- SimplePrograms are synthesized with a combination of brute-force (for programs up to 5 instructions), genetic programming, and MCMC methods.
Experiments

- *Linux Kernel* and *Hutter Prize Wikipedia* datasets are used for evaluation. Metrics used are *bits-per-character* (entropy of $p(x_t|x_{<t})$) and *error rate* (number of mistakes).

- TChar model is compared to various n-gram models (4-, 7-, 10-, and 15-gram) and LSTMs of various sizes.
Experiments

For *Linux Kernel*, TChar model reduces error rate of best baseline (15-gram model) by 35%, reduces BPC by 25%, and is several times faster to train and query than an LSTM!

![Linux Kernel Dataset (Karpathy et al., 2015)]

<table>
<thead>
<tr>
<th>Model</th>
<th>Bits per Character</th>
<th>Error Rate</th>
<th>Training Time</th>
<th>Queries per Second</th>
<th>Model Size</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>LSTM (Layers×Hidden Size)</em></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2×128</td>
<td>2.31</td>
<td>40.1%</td>
<td>≈28 hours</td>
<td>4 000</td>
<td>5 MB</td>
</tr>
<tr>
<td>2×256</td>
<td>2.15</td>
<td>37.9%</td>
<td>≈49 hours</td>
<td>1 100</td>
<td>15 MB</td>
</tr>
<tr>
<td>2×512</td>
<td>2.05</td>
<td>38.1%</td>
<td>≈80 hours</td>
<td>300</td>
<td>53 MB</td>
</tr>
<tr>
<td><em>n-gram</em></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4-gram</td>
<td>2.49</td>
<td>47.4%</td>
<td>1 sec</td>
<td>46 000</td>
<td>2 MB</td>
</tr>
<tr>
<td>7-gram</td>
<td>2.23</td>
<td>37.7%</td>
<td>4 sec</td>
<td>41 000</td>
<td>24 MB</td>
</tr>
<tr>
<td>10-gram</td>
<td>2.32</td>
<td>36.2%</td>
<td>11 sec</td>
<td>32 000</td>
<td>89 MB</td>
</tr>
<tr>
<td>15-gram</td>
<td>2.42</td>
<td>35.9%</td>
<td>23 sec</td>
<td>21 500</td>
<td>283 MB</td>
</tr>
<tr>
<td><em>DSL model (This Work)</em></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TChar w/o cache &amp; backoff</td>
<td>1.92</td>
<td>33.3%</td>
<td>≈8 hours</td>
<td>62 000</td>
<td>17 MB</td>
</tr>
<tr>
<td>TChar w/o backoff</td>
<td>1.84</td>
<td>31.4%</td>
<td>≈8 hours</td>
<td>28 000</td>
<td>19 MB</td>
</tr>
<tr>
<td>TChar w/o cache</td>
<td>1.75</td>
<td>28.0%</td>
<td>≈8.2 hours</td>
<td>24 000</td>
<td>43 MB</td>
</tr>
<tr>
<td>TChar</td>
<td>1.53</td>
<td>23.5%</td>
<td>≈8.2 hours</td>
<td>3 000</td>
<td>45 MB</td>
</tr>
</tbody>
</table>
Experiments

- TChar model is not as good on unstructured data: on Wikipedia, its error rate is roughly the same as for the Linux Kernel dataset, but it is outperformed here by LSTMs.
Advantages

- Program $f$ drawn from TChar can be read by humans; much more interpretable than weights of a neural network.
- Calculating $P(x_t|f(t, x))$ is efficient: use a hashtable to look up how frequently $x$ appears in the context of $f(t, x)$.
- TChar model outperforms LSTMs and n-gram models on structured data.
Disadvantages & Future Work

- TChar model is outperformed by LSTMs on unstructured data.
- TChar has limited expressiveness, unlike DNNs.
- However, increasing the expressiveness of TChar can in theory make the synthesis problem intractable or even undecidable.