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

- 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.

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

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

- 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

$$x = \texttt{Dogs are th}\_{t}$$

- 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!

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).

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).

Learning

- Given a validation set D and regularization penalty Ω, the learning process is to find a program $p^* \in \texttt{TChar}$:

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

- `TChar` consists of branches and `SimplePrograms`.
- Branches are synthesized use 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

Linux Kernel Dataset (Karpathy et al., 2015)					
Model	**Bits per Character**	**Error Rate**	**Training Time**	**Queries per Second**	**Model Size**
LSTM (Layers×Hidden Size)					
2×128	2.31	40.1%	≈28 hours	4 000	5 MB
2×256	2.15	37.9%	≈49 hours	1 100	15 MB
2×512	2.05	38.1%	≈80 hours	300	53 MB
***n*-gram**					
4-gram	2.49	47.4%	1 sec	46 000	2 MB
7-gram	2.23	37.7%	4 sec	41 000	24 MB
10-gram	2.32	36.2%	11 sec	32 000	89 MB
15-gram	2.42	35.9%	23 sec	21 500	283 MB
DSL model (This Work)					
$\texttt{TChar}_{\text{w/o cache \& backoff}}$	1.92	33.3%	≈8 hours	62 000	17 MB
$\texttt{TChar}_{\text{w/o backoff}}$	1.84	31.4%	≈8 hours	28 000	19 MB
$\texttt{TChar}_{\text{w/o cache}}$	1.75	28.0%	≈8.2 hours	24 000	43 MB
`TChar`	1.53	23.5%	≈8.2 hours	3 000	45 MB

- 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!

Experiments

Hutter Prize Wikipedia Dataset (Hutter, 2012)						
Metric	***n*-gram**	**DSL model**	**Stacked LSTM**	**MRNN**	**MI-LSTM**	**HM-LSTM†**
	$n = 7$	This Work	Graves (2013)	Sutskever et al. (2011)	Wu et al. (2016)	Chung et al. (2017)
BPC	1.94	1.62	1.67	1.60	1.44	1.34

- `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.