Program Synthesis

Беляев Станислав
Руководитель: Алексей Шпильман

Prophet

http://groups.csail.mit.edu/pac/patchgen/papers/prophet-popl16.pdf

Идея:

- ищем потенциально плохие места в программе
- изменяем, конечным числом преобразований
- ранжируем патчи, обучая вероятностную модель

Фундаментальная проблема:

- корректного патча с большой вероятностью может не оказаться в генерируемом пространстве патчей
  - преобразований мало
  - сложно добавить новые преобразования (нужно писать код)

Insert Control Flow

S ⟹ if (**E**) return **K**; S

S ⟹ if (**E**) goto **L**; S

S ⟹ if (**E**) break; S

Insert Guard

S ⟹ if (**E**) { S }

Change Condition

if (C) {…} else {…} ⟹ if (C && **E**) {…} else {…}

if (C) {…} else {…} ⟹ if (C || **E**) {…} else {…}

Replace

S ⟹ S[e1/e2]

S ⟹ S[c1/c2]

S ⟹ S[f(e1, …, en)/g(e1,…,en)]

Copy

S ⟹ Q[e1/e2]; S

S ⟹ Q[c1/c2]; S

S ⟹ Q[f(e1, …, en)/g(e1,…,en)]; S

Initialize

S ⟹ memset(&e, 0, sizeof(e)); S

Bayesian encoder & decoder

https://arxiv.org/pdf/1503.00075.pdf

- задача - определить распределение P(X|Θ)
  - X - программа или ее часть
  - Θ - evidence, например, последовательность методов

$$P(X|\Theta) = \int P(X|\Psi)P(\Psi|\Theta)d\Psi$$

- делим задачу на две - P(Ψ|Θ) (encoder) и P(X|Ψ) (decoder)
  - Ψ - латентный вектор, в котором заключен “смысл программы”
- encoder собирает смысл по программе
  - идея: перейти от программе как последовательности токенов к программе как дереву ast
- decoder итеративно генерирует новую программу
  - как-бы разворачивает собранный смысл в новую программу
  - начинаем с пустого ast, генерируем ребенка, а дальше его братьев
- проблемы
  - что делать с литералами при генерации?
  - можно ли абстрагироваться от языка?

Encoder

https://arxiv.org/pdf/1503.00075.pdf

- программа = слово из КС грамматики = дерево ast
  - парсинг реализован как форк pycparser
- векторизация word2vec со связью родитель-ребенок
  - реализована на tensorflow
- child sum tree LSTM
  - проходимся от листьев к корню, пересчитывая вектора внутренних вершин
  - вектор корня и есть вектор, в котором собран весь “смысл” дерева

$$\tilde{h}_j = \sum_{k \in C(j)} h_k,$$

$$i_j = \sigma\left(W^{(i)} x_j + U^{(i)} \tilde{h}_j + b^{(i)}\right),$$

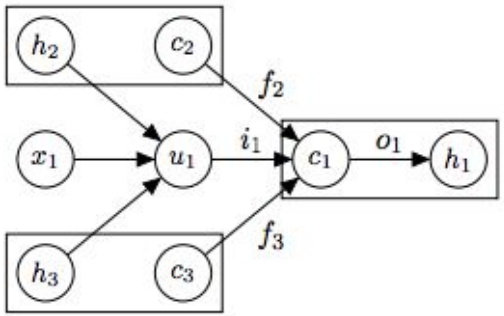
$$f_{jk} = \sigma\left(W^{(f)} x_j + U^{(f)} h_k + b^{(f)}\right),$$

$$o_j = \sigma\left(W^{(o)} x_j + U^{(o)} \tilde{h}_j + b^{(o)}\right),$$

$$u_j = \tanh\left(W^{(u)} x_j + U^{(u)} \tilde{h}_j + b^{(u)}\right)$$

$$c_j = i_j \odot u_j + \sum_{k \in C(j)} f_{jk} \odot c_k,$$

$$h_j = o_j \odot \tanh(c_j),$$



Semantic relatedness

- хорошо работает для естественных языков и для синтаксических деревьев
- есть интуиция, что также будет хорошо работать и для ast
- К степеней похожести, по возрастанию
- KL - мера удаленности распределений
- обучающая выбора - посылки задачек с codeforces.ru

$$h_\times = h_L \odot h_R,$$
$$h_+ = |h_L - h_R|,$$
$$h_s = \sigma\left(W^{(\times)} h_\times + W^{(+)} h_+ + b^{(h)}\right),$$
$$\hat{p}_\theta = \mathrm{softmax}\left(W^{(p)} h_s + b^{(p)}\right),$$
$$\hat{y} = r^T \hat{p}_\theta,$$
$$r^T = [1\ 2\ \ldots\ K]$$

$$J(\theta) = \frac{1}{m} \sum_{k=1}^{m} \mathrm{KL}\left(p^{(k)} \,\middle\|\, \hat{p}_\theta^{(k)}\right) + \frac{\lambda}{2} \|\theta\|_2^2$$

accuracy ≈ 0.82

Результаты

- изучил предметную область
- реализована encoder-часть
  - парсинг
  - векторизация
  - tree LSTM
- протестирована на задаче semantic-relatedness для программ
  - 0.82
- использовал
  - python (theano, tensorflow, pycparser, pickle)

Спасибо за внимание

https://github.com/StasBel/program-synthesis