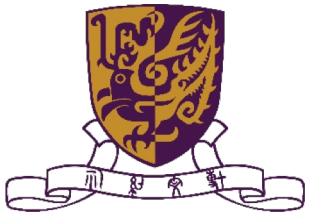
Code Completion with Neural Attention and Pointer Networks

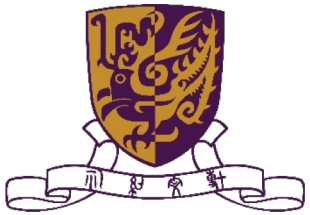
Yue Wang, Jian Li, Michael R. Lyu, Irwin King
{yuewang, jianli, lyu, king}@cse.cuhk.edu.hk

WANG, Yue
02/12/2017



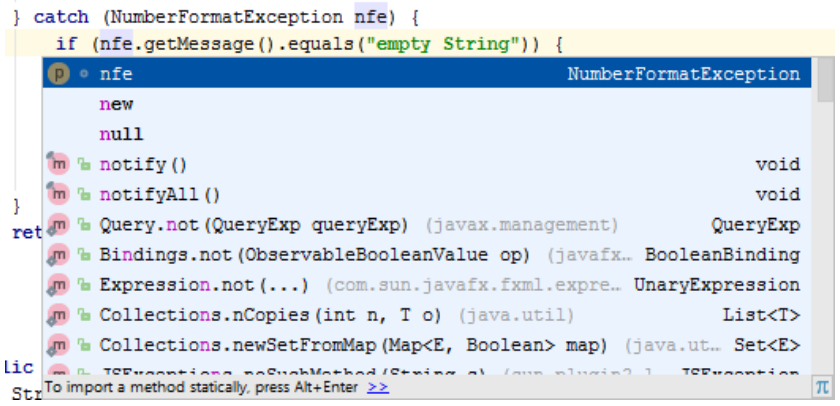
Outlines

- Background
- Motivation
- Contribution
- Approach
- Evaluation
- Discussion
- Conclusion



Background

- Code completion



Harder

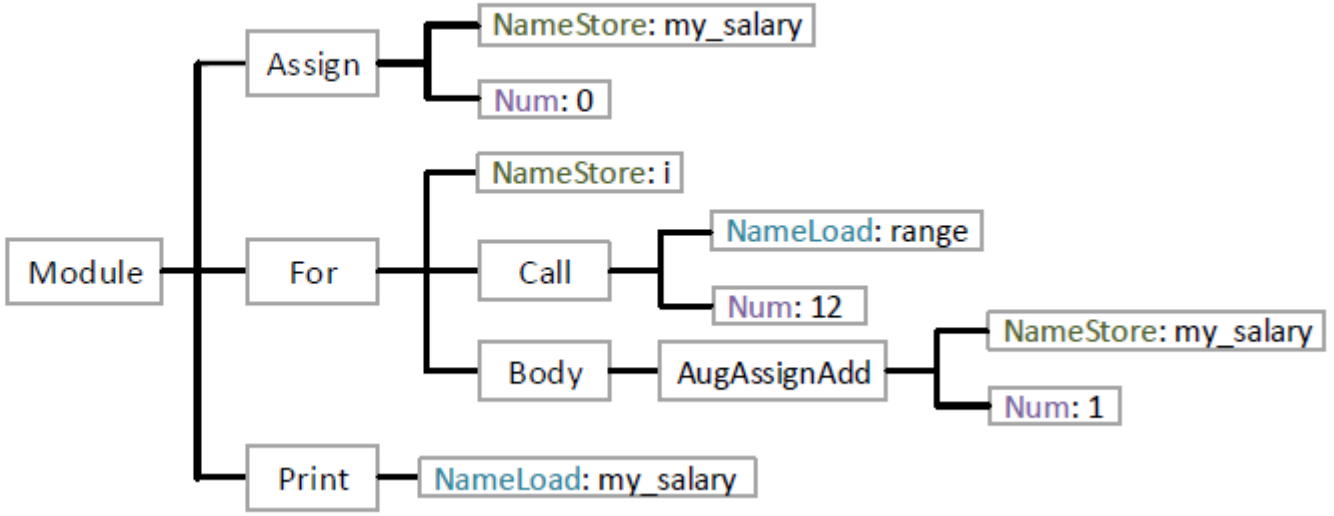
- Programming languages: static vs dynamic
- OoV problem: many words are sparse

Motivation

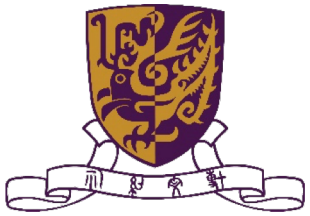
- Abstract syntax tree (AST)
- Locally repeated terms

```
my_salary = 0
for i in range(12):
    my_salary += 1
print(my_salary)
```

CFG

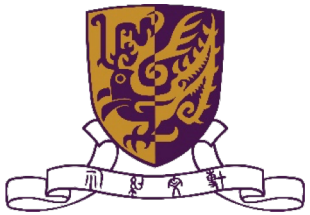


A Python program and its corresponding abstract syntax tree



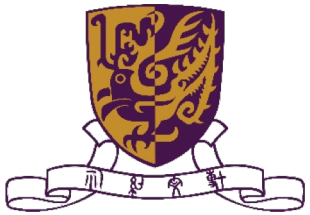
Contribution

- Propose a pointer mixture network for better predicting OoV words in AST-based code completion
- Demonstrate the effectiveness of attention mechanism on code completion
- Our methods show significant improvements upon the state-of-the-art on two benchmarked datasets



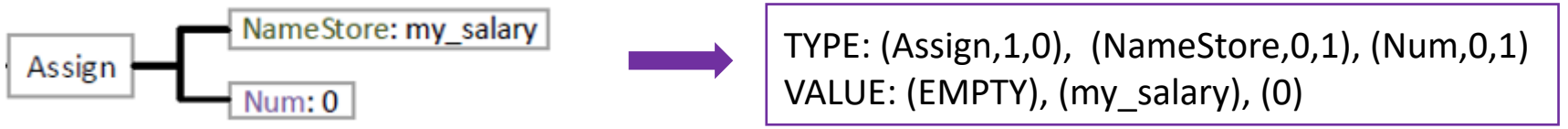
Approach

- Program representation
- Neural language model
- Attention mechanism
- Pointer mixture network



Approach

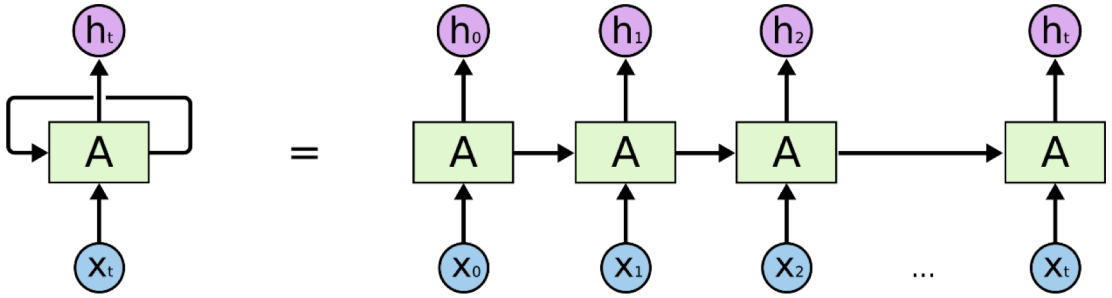
- Program representation
  - Serialize each AST as a sequence of nodes in the in-order depth-first traversal
  - Encode two additional bits of information about whether the AST node has a child and/or a right sibling



- Given a sequence of words $w_1, \ldots, w_{t-1}$, our task is to predict the next word w_t.
  - Build one model for each task: TYPE and VALUE

Approach

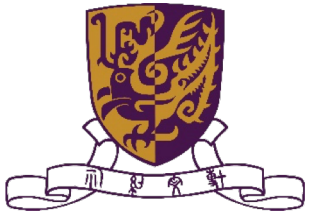
- Neural language model
  - Task formalism: $p(w_t|w_1, w_2, \ldots, w_{t-1}; \theta)$
- Traditions: n-gram
- Advances: Recurrent Neural Networks (RNNs)
  - LSTM is a variant of RNN
  - A LSTM cell: $(h_t, s_t) = f(x_t, h_{t-1}, s_{t-1})$



$$\begin{bmatrix} i_t \\ f_t \\ o_t \\ \tilde{s}_t \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{bmatrix} W_{4k,2k} \begin{bmatrix} x_t \\ h_{t-1} \end{bmatrix}$$

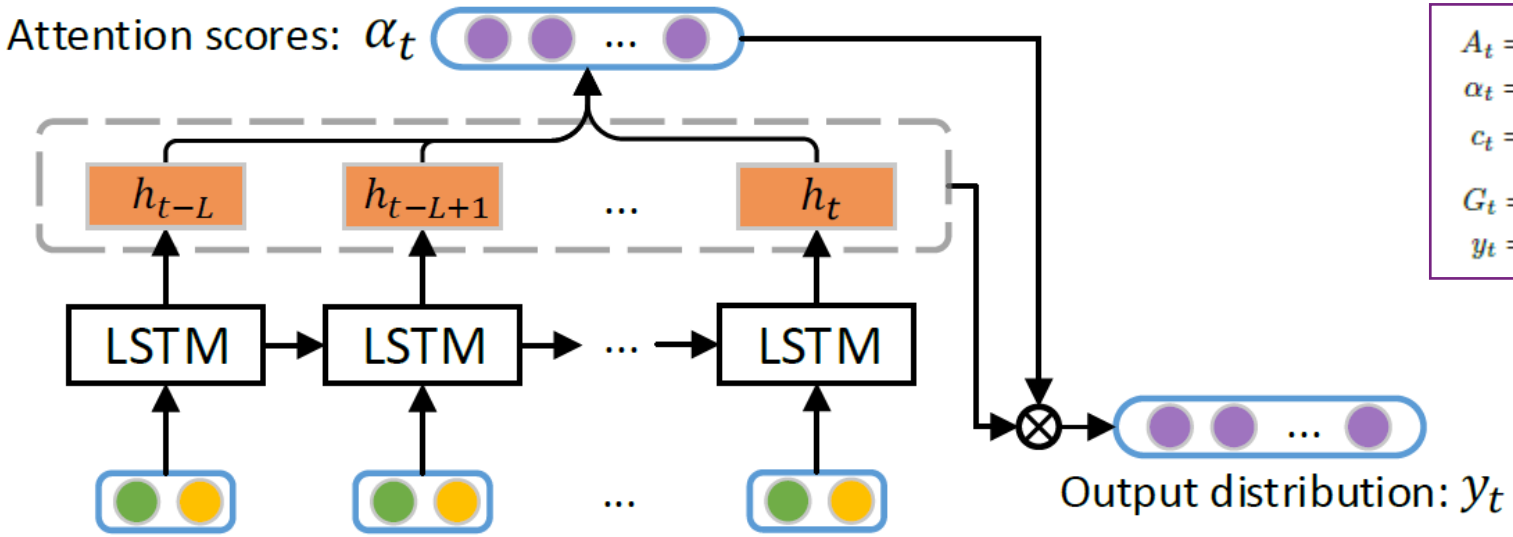
$$s_t = f_t \odot s_{t-1} + i_t \odot \tilde{s}_t$$

$$h_t = o_t \odot \tanh(s_t)$$



Approach

- Attention mechanism

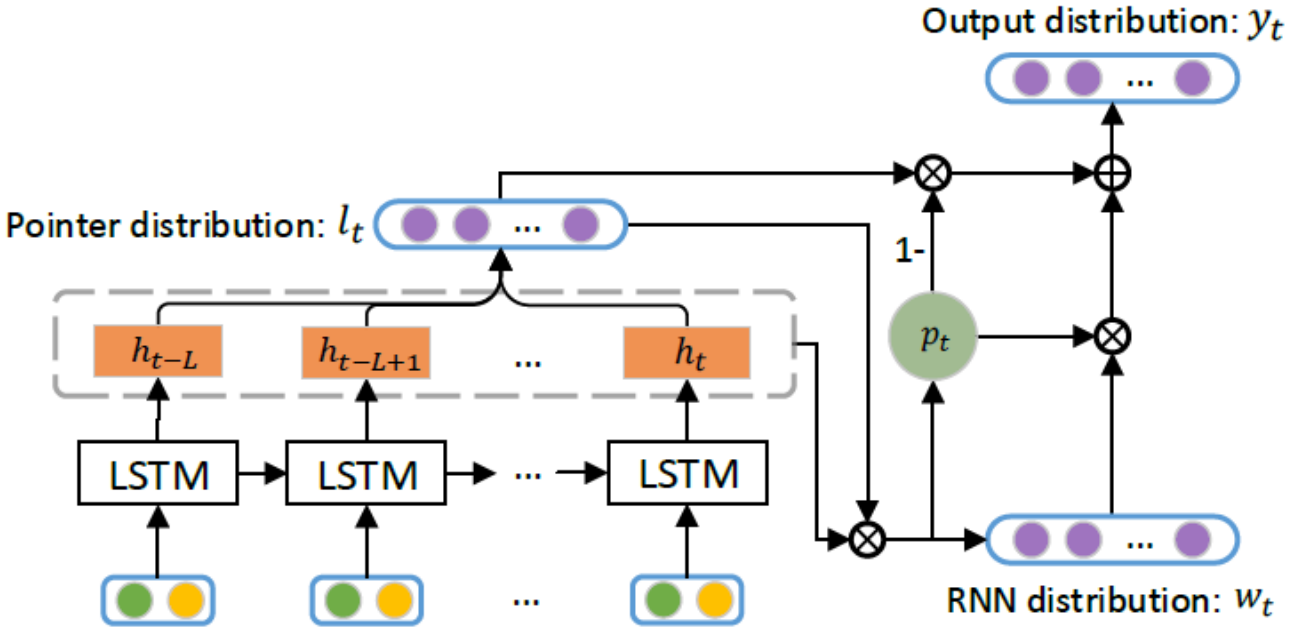


$$A_t = v^T \tanh(W^m M_t + (W^h h_t)1_L^T)$$
$$\alpha_t = softmax(A_t)$$
$$c_t = M_t \alpha_t^T$$
$$G_t = \tanh(W^g [h_t; c_t])$$
$$y_t = softmax(W^v G_t + b^v)$$

- Resolve the hidden state bottleneck

Approach

- Pointer mixture network
  - Global RNN component
  - Local pointer component
  - Controller



$$p_t = \sigma(W^p[h_t; c_t] + b^p)$$

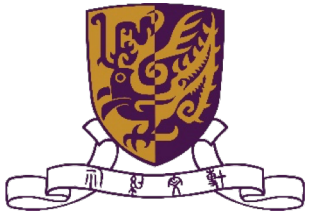
$$y_t = softmax([p_t w_t; (1 - p_t) l_t])$$

Evaluation

- Dataset
  - JavaScript (JS) and Python (PY)

Table 1: Dataset Statistics

	JS	PY
Training Queries	$10.7 * 10^7$	$6.2 * 10^7$
Test Queries	$5.3 * 10^7$	$3.0 * 10^7$
Type Vocabulary	95	329
Value Vocabulary	$2.6 * 10^6$	$3.4 * 10^6$



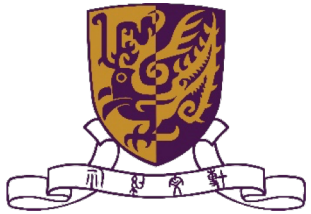
Evaluation

- Accuracies on next value prediction with different vocabulary sizes

Table 2: Accuracies on next value prediction with different vocabulary sizes. The out-of-vocabulary (OoV) rate denotes the percentage of AST nodes whose value is beyond the global vocabulary.

	JS			PY		
Vocabulary Size (OoV Rate)	1k (20%)	10k (11%)	50k (7%)	1k (24%)	10k (16%)	50k (11%)
Vanilla LSTM	69.9%	75.8%	78.6%	63.6%	66.3%	67.3%
Attention-enhanced LSTM (ours)	71.7%	78.1%	80.6%	64.9%	68.4%	69.8%
Pointer Mixture Network (ours)	**73.2%**	**78.9%**	**81.0%**	**66.4%**	**68.9%**	**70.1%**

- Observation:
  - When increasing the vocabulary, the OoV rate decreases obviously, and the general accuracies increase but the performance gain for pointer mixture network decreases
  - Pointer mixture network consistently outperform other two models over different vocabulary size



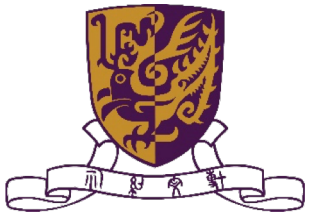
Evaluation

- Comparisons against the state-of-the-arts
  - Note that Pointer Mixture Network can be only used for predicting VALUE node (TYPE node has small size of vocabulary)

Table 3: Comparisons against the state-of-the-arts. The upper part is the results from our experiments while the lower part is the results from prior work. TYPE means next node type prediction and VALUE means next node value prediction.

	JS		PY	
	TYPE	VALUE	TYPE	VALUE
Vanilla LSTM	87.1%	78.6%	79.3%	67.3%
Attention-enhanced LSTM (ours)	**88.6%**	80.6%	**80.6%**	69.8%
Pointer Mixture Network (ours)	-	81.0%	-	**70.1%**
LSTM (Liu et al. 2016)	84.8%	76.6%	-	-
Probabilistic Model (Raychev et al. 2016)	83.9%	**82.9%**	76.3%	69.2%

- Observations: our models outperform the state-of-the-art in almost all cases

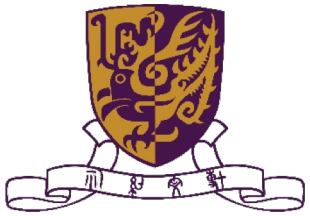


Discussion

- Why pointer mixture network works?
  - Pointer Random Network: randomly generate the weights for the pointer component, while the rest is the same as the Pointer Mixture Network

Table 4: Showing why pointer mixture network works.

	JS_1k	PY_1k
Pointer Random Network	71.4%	64.8%
Attention-enhanced LSTM	71.7%	64.9%
Pointer Mixture Network	**73.2%**	**66.4%**



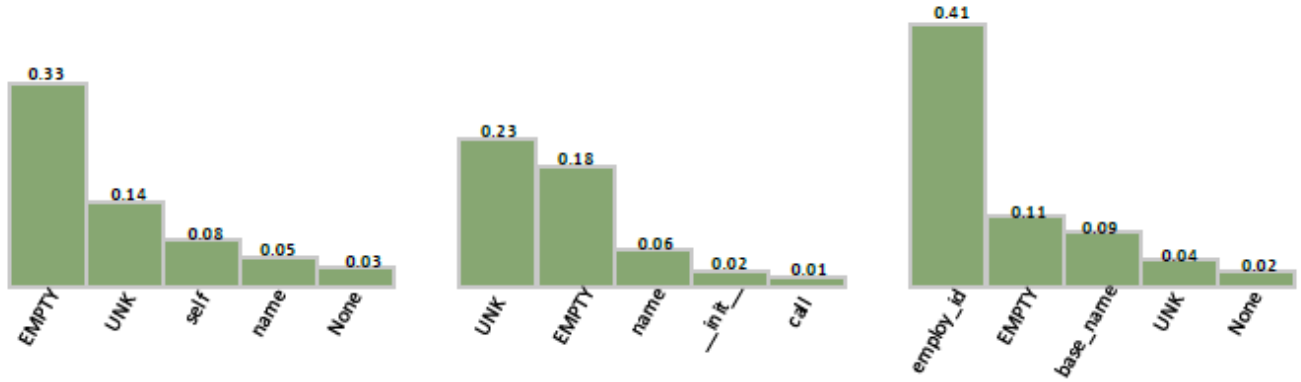
Discussion

- Case study

```
class Operator(Employee):
  def __init__(self, name, employee_id):
    super(Operator, self).__init__(name, Rank.OPERATOR)
    self.employee_id = employee_id

  def _dispatch_call(self, call, employees):
    for employee in employees:
      employee.take_call(call)

  def record_path(self, base_name):
    return os.path.join(base_name, str(self.___?___))
```

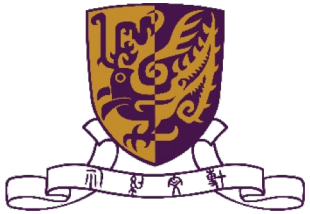


(a) Vanilla LSTM (b) Attention-enhanced LSTM (c) Pointer Mixture Network

Conclusion

- First demonstrate the effectiveness of attention mechanism on AST-based code completion
- Propose a pointer mixture network which learns to either generate a new value or copy an OoV value from local context

- Future work:
  - Incorporate more static type information of programs
  - Extend our models towards automated code generation



Thanks for listening !