Generative Code Modeling With Graphs

MarcBrockschmidt, MiltiadisAllamanis, AlexanderGaunt, OleksandrPolozov

From Microsoft

ICLR-2019

Main Tasks

- Typical:

  a line of code

- New Task: **more logic**

  part of code

```
int ilOffsetIdx =
   Array.IndexOf(sortedILOffsets, map.ILOffset);
int nextILOffsetIdx = ilOffsetIdx + 1;
int nextMapILOffset =
   nextILOffsetIdx < sortedILOffsets.Length
   ? sortedILOffsets[nextILOffsetIdx]
   : int.MaxValue;
```

How to Judge?

- Perplexity
  - H(p) =-Σ p(x) log p(x)
  - Perplecity = 2 ^ {H(p)}

$$perplexity(S) = p(w_1, w_2, w_3, \ldots, w_m)^{-1/m}$$
$$= \sqrt[m]{\prod_{i=1}^{m} \frac{1}{p(w_i | w_1, w_2, \ldots, w_{i-1})}}$$

- Beam Search
  - Expand the most promising node
  - Reduce memory requirements

Prerequisite

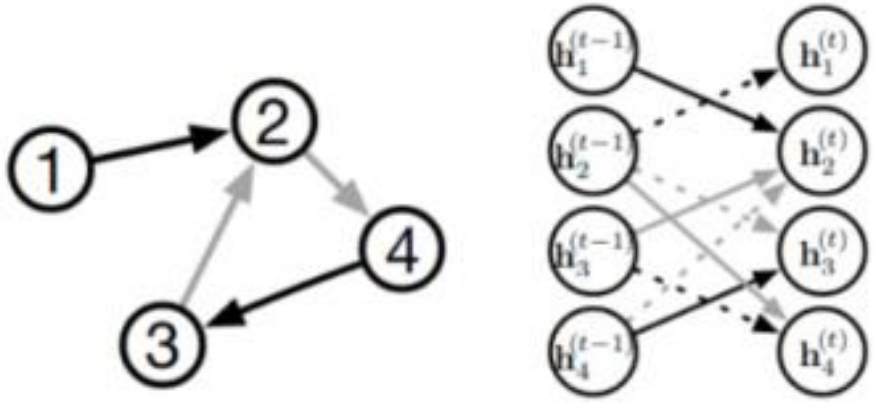
- Embedding
  - Dense vector of the same length → dimensional reduction
- Attribute Grammar (context-free grammar)
  - Inherited
  - Synthesized
- Graph Neural Network
- Encoder-Decoder
- Attention

$$\mathbf{Expr}_1 \rightarrow \mathbf{Expr}_2 + \mathbf{Term} \ [\ \mathbf{Expr}_1.\text{value} = \mathbf{Expr}_2.\text{value} + \mathbf{Term}.\text{value}\ ]$$
$$\mathbf{Expr} \rightarrow \mathbf{Term} \ [\ \mathbf{Expr}.\text{value} = \mathbf{Term}.\text{value}\ ]$$
$$\mathbf{Term}_1 \rightarrow \mathbf{Term}_2 * \mathbf{Factor} \ [\ \mathbf{Term}_1.\text{value} = \mathbf{Term}_2.\text{value} * \mathbf{Factor}.\text{value}\ ]$$
$$\mathbf{Term} \rightarrow \mathbf{Factor} \ [\ \mathbf{Term}.\text{value} = \mathbf{Factor}.\text{value}\ ]$$
$$\mathbf{Factor} \rightarrow \text{“(“}\ \mathbf{Expr}\ \text{“)”} \ [\ \mathbf{Factor}.\text{value} = \mathbf{Expr}.\text{value}\ ]$$
$$\mathbf{Factor} \rightarrow \mathit{integer} \ [\ \mathbf{Factor}.\text{value} = \text{strToInt}(\mathit{integer}.\text{str})\ ]$$

Prerequisite: GNN

- semi-supervised

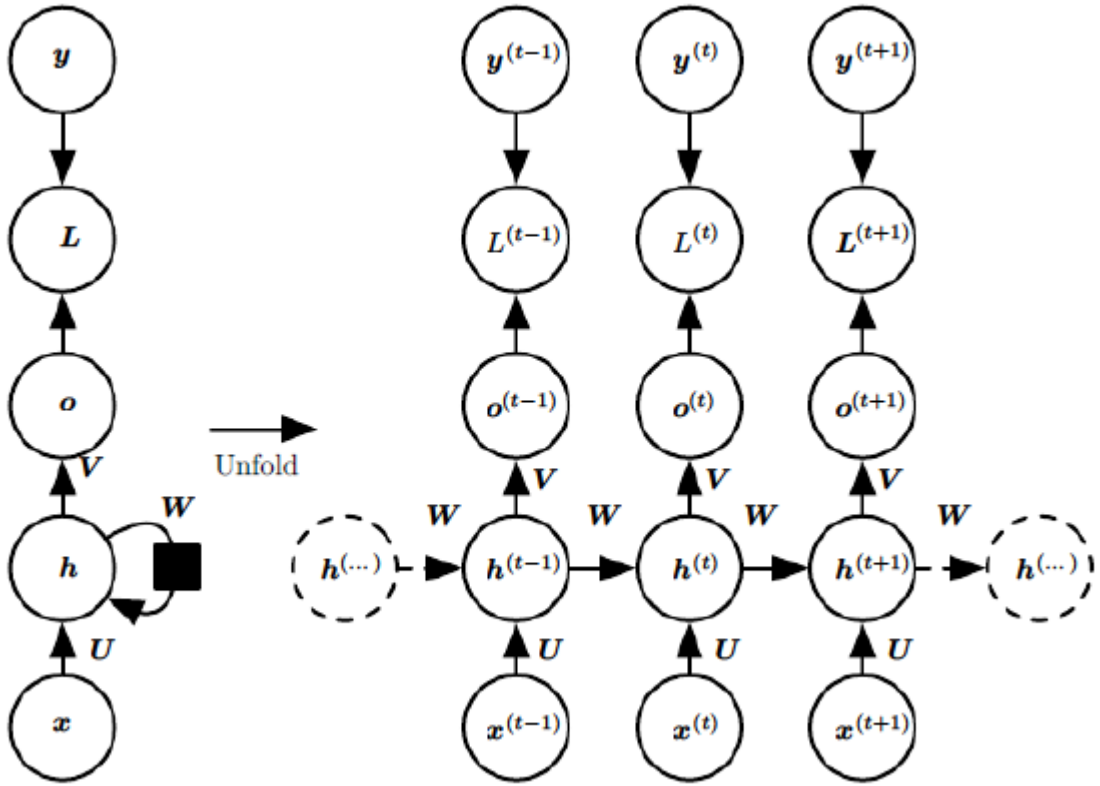
- message passing
- neighborhood aggregation



- Deep walk – embedding (KDD' 14)
  - Random sequence
  - Skip-gram (stride: 1)

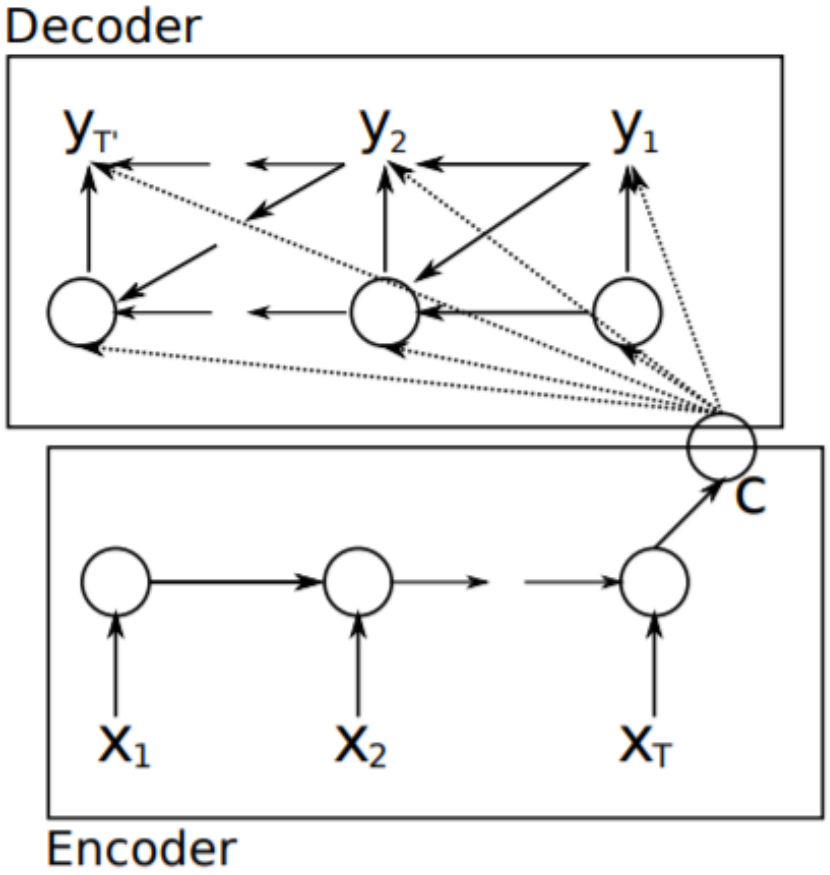
Prerequisite: RNN

- Influenced by what it has learnt from the past
- Learnt from prior input while generating output
- Bidirectional RNNs
  - Know about the past and the future



Prerequisite: Encoder - Decoder

- Encoder
  - A stack of several recurrent units
- Intermediate vector
  - Initial hidden state of the decoder part of the model
- Decoder
  - accepts a hidden state from the previous unit
  - produces and output as well as its own hidden state

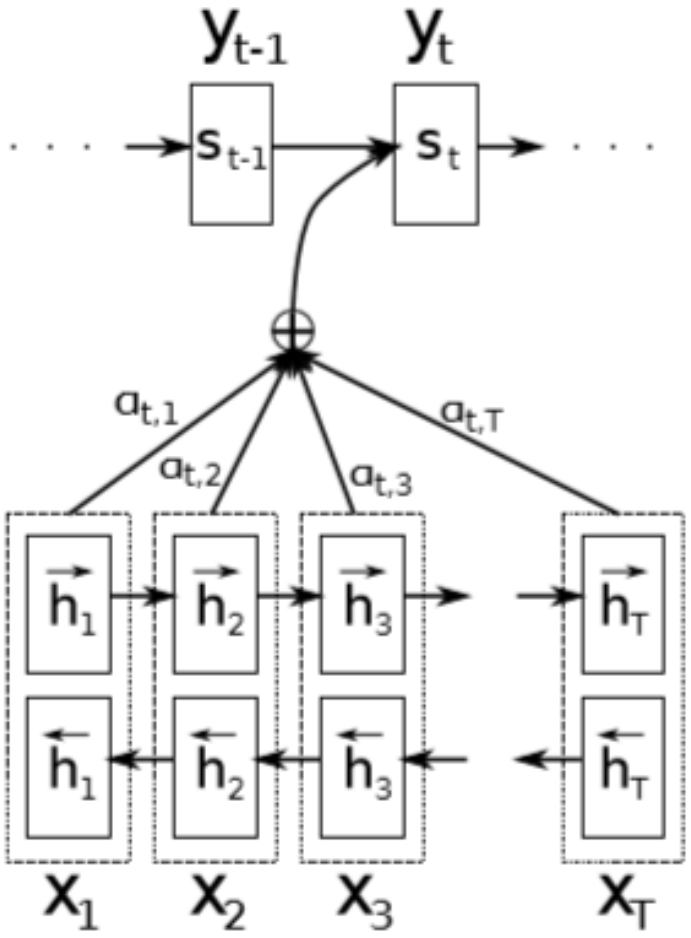


$$p(a|c) = \prod_t p(a_t|c, a_{<t})$$

Prerequisite: Attention

- Why attention is needed?
  - fixed-length context vector
  - forget the earlier parts of the sequence

- How it work?
  - Compute a score each encoder state
  - Compute the attention weights
  - Compute the context vector
  - Concatenate context vector with output of previous time step
  - Decoder Output



Related Works

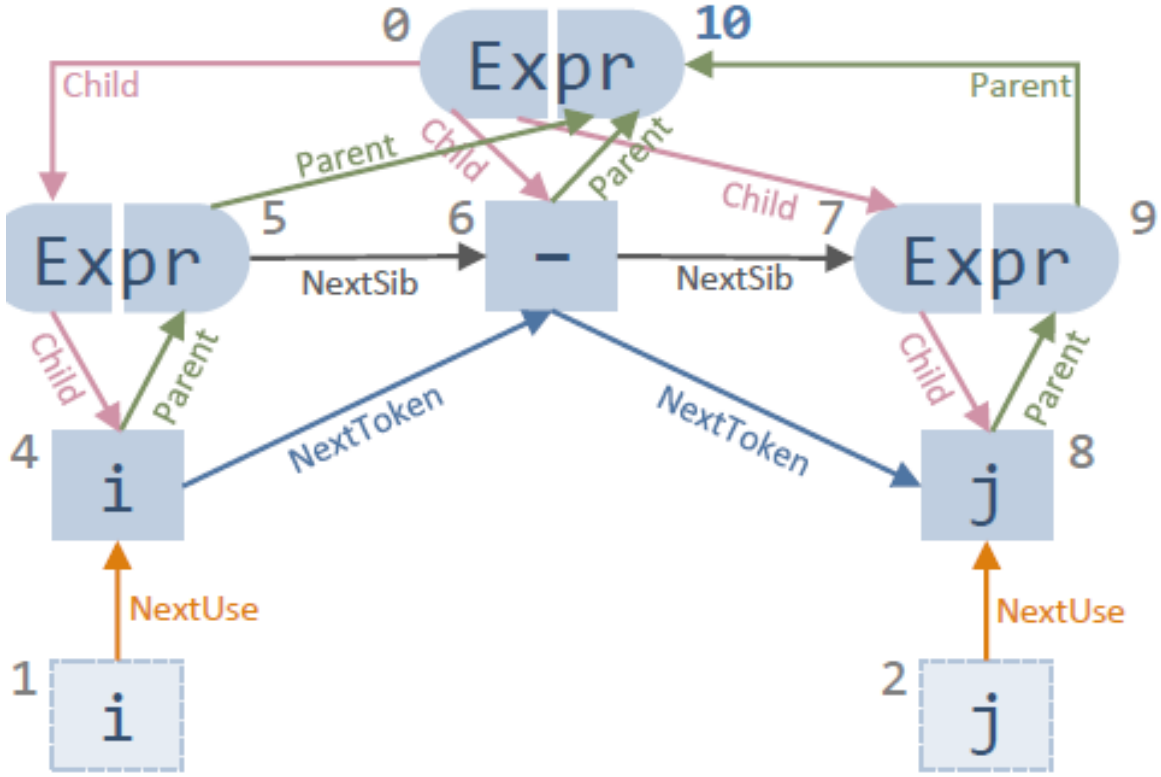
- Natural Language Processing
  - Can not distinguish unlikely from likely
  - Fail to be semantically relevant
  - Wrong on human inspection
- Abstract Syntax Tree
  - Costly on calculation resources
  - Modify model to fit the AST

→ New method: sequence

```
expression: BinaryExpression  {
   type: "BinaryExpression"
   start: 0
   end: 7
 - left: BinaryExpression  {
      type: "BinaryExpression"
      start: 1
      end: 4
    + left: Literal {type, start, end, value, raw}
      operator: "+"
    + right: Literal {type, start, end, value, raw}
   }
   operator: "*"
 - right: Literal  {
      type: "Literal"
      start: 6
      end: 7
      value: 3
      raw: "3"
   }
}
```

Overview

- Build basic AST and graph
- Get context information & representation of root node
- Keep expanding node
- Update (MLE)



Expand Node

- Common generation strategy
- Expand AST & graph

$$\mathbf{h}_v = g(\text{emb}(\ell_v), \sum_{\langle u_i, t_i, v\rangle \in \mathcal{E}_v} f_{t_i}(\mathbf{h}_{u_i}))$$

Algorithm 1 Pseudocode for Expand

Input: Context c, partial AST a, node v to expand

$\mathbf{h}_v \leftarrow \text{getRepresentation}(c, a, v)$
$rhs \leftarrow \text{pickProduction}(v, \mathbf{h}_v)$
for child node type $\ell \in rhs$ **do**
  $(a, u) \leftarrow \text{insertChild}(a, \ell)$
  if ℓ is nonterminal type **then**
    $a \leftarrow \text{Expand}(c, a, u)$
return a

Edges

- Typed directed edges

- Embed different edge type into different space

Algorithm 2 Pseudocode for ComputeEdge

Input: Partial AST a, node v

Edge set $\mathcal{E} \leftarrow \varnothing$
if v is inherited **then**
  $\mathcal{E} \leftarrow \mathcal{E} \cup \{\langle \mathsf{parent}(a, v), \mathit{Child}, v\rangle\}$
  if v is terminal node **then**
    $\mathcal{E} \leftarrow \mathcal{E} \cup \{\langle \mathsf{lastToken}(a, v), \mathit{NextToken}, v\rangle\}$
    if v is variable **then**
      $\mathcal{E} \leftarrow \mathcal{E} \cup \{\langle \mathsf{lastUse}(a, v), \mathit{NextUse}, v\rangle\}$
  if v is not first child **then**
    $\mathcal{E} \leftarrow \mathcal{E} \cup \{\langle \mathsf{lastSibling}(a, v), \mathit{NextSib}, v\rangle\}$
else
  $\mathcal{E} \leftarrow \mathcal{E} \cup \{\langle u, \mathit{Parent}, v\rangle \mid u \in \mathsf{children}(a, v)\}$
  $\mathcal{E} \leftarrow \mathcal{E} \cup \{\langle \mathsf{inheritedAttr}(v), \mathit{InhToSyn}, v\rangle\}$
return $\mathcal{E}$

pickProduction

- choose production → classification
- $pickProduction = argmax\ P(rule|l_v, h_v)$

pickVariable

- Pointer Network
  - sequence-to-sequence & attention
  - output of pointer networks is discrete and correspond to positions in the input sequence
  - the number of target classes in each step of the output depends on the length of the input

- $\text{pick}Variable(\mathbb{\Gamma}, h_v) = argmax\ P(var|h_v)$

pickLiteral

- Copy one of the tokens
- For each token compute a score
- Only way to generate a UNK token

- $pickLiteral(token, h_v) = argmax\ p(lit|h_v)$

Results

Model	Test (from seen projects)				Test-only (from unseen projects)			
	Perplexity	Well-Typed	Acc@1	Acc@5	Perplexity	Well-Typed	Acc@1	Acc@5
PHOG†	–	–	*34.8%*	*42.9%*	–	–	*28.0%*	*37.3%*
$\mathcal{S}eq \rightarrow \mathcal{S}eq$	87.48	32.4%	21.8%	28.1%	130.46	23.4%	10.8%	16.8%
$\mathcal{S}eq \rightarrow \mathcal{NAG}$	6.81	53.2%	17.7%	33.7%	8.38	40.4%	8.4%	15.8%
$\mathcal{G} \rightarrow \mathcal{S}eq$	93.31	40.9%	27.1%	34.8%	28.48	36.3%	17.2%	25.6%
$\mathcal{G} \rightarrow \mathcal{T}ree$	4.37	49.3%	26.8%	48.9%	5.37	41.2%	19.9%	36.8%
$\mathcal{G} \rightarrow \mathcal{ASN}$	2.62	78.7%	45.7%	62.0%	3.03	74.7%	32.4%	48.1%
$\mathcal{G} \rightarrow \mathcal{S}yn$	2.71	84.9%	50.5%	66.8%	3.48	84.5%	36.0%	52.7%
$\mathcal{G} \rightarrow \mathcal{NAG}$	2.56	86.4%	52.3%	69.2%	3.07	84.5%	38.8%	57.0%

```
int methParamCount = 0;
if (paramCount > 0) {
 IParameterTypeInformation[] moduleParamArr =
  GetParamTypeInformations(Dummy.Signature, paramCount);
 methParamCount = moduleParamArr.Length;
}
if (paramCount > methParamCount) {
 IParameterTypeInformation[] moduleParamArr =
  GetParamTypeInformations(Dummy.Signature,
                           paramCount - methParamCount);
}
```

$\mathcal{G} \rightarrow \mathcal{NAG}$:
`paramCount > methParamCount` (34.4%)
`paramCount == methParamCount` (11.4%)
`paramCount < methParamCount` (10.0%)

$\mathcal{G} \rightarrow \mathcal{ASN}$:
`paramCount == 0` (12.7%)
`paramCount < 0` (11.5%)
`paramCount > 0` (8.0%)

```
public static String URItoPath(String uri) {
   if (System.Text.RegularExpressions
         .Regex.IsMatch(uri, "^file:\\\\[a-z,A-Z]:")) {
      return uri.Substring(6);
   }
   if (uri.StartsWith(@"file:")) {
      return uri.Substring(5);
   }
   return uri;
}
```

$\mathcal{G} \rightarrow \mathcal{NAG}$:
`uri.Contains(UNK_STRING_LITERAL)` (32.4%)
`uri.StartsWith(UNK_STRING_LITERAL)` (29.2%)
`uri.HasValue()` (7.7%)

$\mathcal{G} \rightarrow \mathcal{S}yn$:
`uri == UNK_STRING_LITERAL` (26.4%)
`uri == ""` (8.5%)
`uri.StartsWith(UNK_STRING_LITERAL)` (6.7%)

Limitations

- Limited types
- Training data noise
- Bad performance on new project

Prospect

- Unified evaluation index
- Qualified dataset

Thanks