Analyzing Software using Deep Learning

Lecture 2: RNN-based Code Completion and Repair

Prof. Dr. Michael Pradel
Software Lab, TU Darmstadt
Plan for Today

- **Deep learning basics**
  - Finish up last lecture

- **Recurrent neural networks (RNNs)**

- **Code completion with statistical language models**
  
  Based on PLDI 2014 paper by Raychev et al.

- **Repair of syntax errors**
  
  Based on "Automated correction for syntax errors in programming assignments using recurrent neural networks" by Bhatia & Singh, 2016
From Neurons to Layers

For every neuron:
output = f(w \cdot x + b)

x, f, b ... scalars, e.g. in \mathbb{R}
w ... vector, e.g., in \mathbb{R}^n

For each layer:
output = f(W \cdot x + b)

x, f, b ... vector, e.g., in \mathbb{R}^n
W ... matrix, e.g., in \mathbb{R}^{m \times n}
Feedforward networks

\[ x \xrightarrow{U} h \xrightarrow{V} y \]

x, h, y - input layer, hidden layer, output layer

U, V, W - weight matrices

\[ \rightarrow \] function

\[ \xrightarrow{\text{delay of single time step}} \] function with delay of single time step

Recurrent networks

\[ x \xrightarrow{U} h \xrightarrow{V} y \]

\[ x \xrightarrow{U} h \xrightarrow{V} y \]

useful for representing sequences of inputs & outputs

store information about previous inputs
Example: Predict next word in sentence

ASDL is the best ... (course)

Feedforward

\[ \text{ASDL} \rightarrow \text{is} \rightarrow \text{best} \rightarrow \text{...} \]

Recurrent

\[ \text{time}=1 \]

\[ \text{ASDL} \rightarrow \text{is} \rightarrow \text{best} \rightarrow \text{course} \]

Recurrent connection remember the beginning of the sentence
Unfolding the Computational Graph

\[ y \quad \xrightarrow{\text{unfold}} \quad h \quad \xrightarrow{\text{best}} \quad y_{t+1} \]

\[ h^t = f(h^{t-1}, x^t) \quad \text{is} \quad h^t = \tanh(W \cdot h^{t-1} + U \cdot x^t + b) \]

\[ y^t = f(h^t) \quad \text{e.g.} \quad y^t = \text{softmax}(V \cdot h^t + c) \]
Softmax Function

- **Goal:** Interpret output vector as a probability distribution

- "Squashes" vector of $k$ values $\in \mathbb{R}$ into vector of $k$ values $\in [0, 1]$ that sum up to 1

- **Definition:**
  \[
  \sigma(y)_j = \frac{e^{y_j}}{\sum_{i=1}^{k} e^{y_i}} \quad \text{for } j = 1, .., k
  \]

- **Example:**
  \[
  \sigma([1, 2, 3, 4, 1, 2, 3]) = [0.024, 0.064, 0.175, 0.475, 0.024, 0.064, 0.175]
  \]
Quiz

Which of the following vectors may be the output of the softmax function?

1.) \( y = [0.0, 0.0, 0.0, 0.0] \)

2.) \( y = [0.0, 0.25, 0.25, 0.5] \)

3.) \( y = [0.0, 1.0, 0.0, 0.0] \)

4.) \( y = [0.1, 0.1, 0.2, 0.3] \)
Quiz

Which of the following vectors may be the output of the softmax function?

1.) \( y = [0.0, 0.0, 0.0, 0.0] \)  sum is not 1
2.) \( y = [0.0, 0.25, 0.25, 0.5] \)
3.) \( y = [0.0, 1.0, 0.0, 0.0] \)
4.) \( y = [0.1, 0.1, 0.2, 0.3] \)  sum is not 1
Applications of RNNs

Useful for tasks where the input (and maybe also the output) is a sequence

Examples:

- Unsegmented connected handwriting recognition
- Machine translation of natural languages
- Video classification by frames
- Speech recognition
- Sentiment analysis of twitter messages
Plan for Today

- Deep learning basics
  - Finish up last lecture
- Recurrent neural networks (RNNs)
- Code completion with statistical language models
  Based on PLDI 2014 paper by Raychev et al.
- Repair of syntax errors
  Based on ”Automated correction for syntax errors in programming assignments using recurrent neural networks” by Bhatia & Singh, 2016
Code Completion

- Given: Partial program with one or more holes
- Goal: Find suitable code to fill into the holes
- Basic variants in most IDEs
- Here: Fill holes with sequences of method calls
  - Which methods to call
  - Which arguments to pass
Example

```java
SmsManager smsMgr = SmsManager.getDefault();
int length = message.length();
if (length > MAX_SMS_MESSAGE_LENGTH) {
    ArrayList<String> msgList =
        smsMgr.divideMsg(message);
    // hole H1
} else {
    // hole H2
}
```
**Statistical Language Models**

- Dictionary of words
- Sentences: sequences of words
- Model: Probability distribution over all possible sentences

**Example:** English

\[ \text{Pr("hello world") > Pr("world hello")} \]

- Most basic model: Predict next word based on all previous words

\[ \text{Pr}(s) = \prod_{i=1}^{m} \text{Pr}(w_i \mid h_{i-1}) \quad \text{where} \quad s = w_1, \ldots, w_m \]

\[ h_i = w_{n-i-1} \ldots w_i. \]
Model-based Code Completion

- Program code $\approx$ *sentences* in a language
- Code completion $\approx$ Finding the *most likely completion* of the current sentence
Model-based Code Completion

- Program code \(\approx\) **sentences** in a language
- Code completion \(\approx\) Finding the **most likely completion** of the current sentence

Challenges

- How to abstract code into sentences?
- What kind of language model to use?
- How to efficiently predict a completion
Overview of SLANG Approach

From "Code Completion with Statistical Language Models" by Raychev et al., 2014
**n-gram Language Model**

Problem of "all history" model: Training data may not contain anything about $h_i$

Idea: Next word depends on $n-1$ previous words

$$Pr(s) = \prod_{i=1}^{n} Pr(w_i | w_{i-(n-1)} \cdot \ldots \cdot w_{i-1})$$

Example: $Pr(\text{to \cdot be} \cdot \text{or} \cdot \text{not} \cdot \text{to} \cdot \text{be})$

$$= Pr(\text{to} | \text{E}) \cdot Pr(\text{be} | \text{to}) \cdot Pr(\text{or} | \text{to} \cdot \text{be})$$

$$\ldots \cdot Pr(\text{be} | \text{not} \cdot \text{to}) \ldots$$

Probability of n-grams: Estimated from corpus of training examples
RNN-based Model

store information about (all) previous words

Encoded as vector: One-hot encoding

- Length = size of dictionary
- All values are zero except position of word set to one
Sequences of Method Calls

Abstracting code into sentences

- Method call \( \approx \) word
- Sequence of method calls \( \approx \) sentence
- Separate sequences for each object
- Objects can occur in call as base object, argument, or return value
Option 1: Dynamic Analysis

Execute program and observe each method call

**Advantage:**
- Precise results

**Disadvantage:**
- Only analyzes executed code

```java
if (getInput() > 5) {
    // Suppose always taken
    obj.foo(); // in analyzed execution
} else {
    obj.bar(); // Never gets analyzed
}
```
Option 2: Static Analysis

Reason about execution without executing the code

Advantage:
- Can consider all execution paths

Disadvantage:
- Need to abstract and approximate actual execution

```java
if (getInput() > 5) {
    a.foo(); // Does this call ever get executed?
}

b.bar(); // May a and b point to the same object?
```
Over- & Underapproximation

Program $P$, Input $i$, Behavior $P(i)$

All possible behaviors (what we want to analyze, ideally)

Underapproximation (most dynamic analyses)

Overapproximation (most static analyses)
Static Analysis of Call Sequences

SLANG approach: Static analysis

- **Bound** the number of analyzed loop iterations
- On control flow joins, take union of possible execution sequences
- **Points-to analysis** to reason about references to objects
Example

SmsManager smsMgr = SmsManager.getDefault();
int length = message.length();
if (length > MAX_SMS_MESSAGE_LENGTH) {
    ArrayList<String> msgList =
        smsMgr.divideMsg(message);
} else {}
Example

SmsManager smsMgr = SmsManager.getDefault();
int length = message.length();
if (length > MAX_SMS_MESSAGE_LENGTH) {
    ArrayList<String> msgList =
        smsMgr.divideMsg(message);
} else {}

5 sequences:

<table>
<thead>
<tr>
<th>Object</th>
<th>Calls</th>
</tr>
</thead>
<tbody>
<tr>
<td>smsMgr</td>
<td>(getDefault, ret)</td>
</tr>
<tr>
<td>smsMgr</td>
<td>(getDefault, ret) · (divideMsg, 0)</td>
</tr>
<tr>
<td>message</td>
<td>(length, 0)</td>
</tr>
<tr>
<td>message</td>
<td>(length, 0) · (divideMsg, 1)</td>
</tr>
<tr>
<td>msgList</td>
<td>(divideMsg, ret)</td>
</tr>
</tbody>
</table>
Training Phase

- Training data used for paper:
  3 million methods from various Android projects
- Extract sentences via static analysis
- Train statistical language model
  - Both n-gram and RNN model
Query Phase

- Given: Method with holes

- For each hole:
  - Consider all possible completions of the partial call sequence
  - Query language model to obtain probability
    - Average of n-gram and RNN models

- Return completed code that maximizes overall probability
SmsManager smsMgr = SmsManager.getDefault();
int length = message.length();
if (length > MAX_SMS_MESSAGE_LENGTH) {
    ArrayList<String> msgList =
        smsMgr.divideMsg(message);
    // hole H1
} else {
    // hole H2
}
Example

SmsManager smsMgr = SmsManager.getDefault();
int length = message.length();
if (length > MAX_SMS_MESSAGE_LENGTH) {
    ArrayList<String> msgList =
        smsMgr.divideMsg(message);
    smsMgr.sendMultipartTextMessage(..., msgList, ...);
} else {
    smsMgr.sendTextMessage(..., message, ...);
}
Scalability Tricks

Search space of possible completions: Too large to explore in reasonable time

Refinements to reduce space

- Users may provide hints
  - How many calls to insert
  - Which objects to use
- Replace infrequent words with "unknown"
- Obtain candidate calls using bi-gram model
- Query language model only for candidates
Plan for Today

- Deep learning basics
  - Finish up last lecture
- Recurrent neural networks (RNNs)
- Code completion with statistical language models
  Based on PLDI 2014 paper by Raychev et al.
- Repair of syntax errors
  Based on "Automated correction for syntax errors in programming assignments using recurrent neural networks" by Bhatia & Singh, 2016