Reliable and Interpretable Artificial Intelligence

Martin Vechev
ETH Zurich

Fall 2017
So far we looked at “program synthesis in the small”, meaning we did not leverage knowledge of prior, similar programs, when synthesizing a new program. In this lecture, we are going to study both directions:

**Part I:** Leverage ML models to enable program synthesis. This will involve learning from a new kind of data sets (e.g., now programs themselves are examples). Motivated by this we will move to **Part II**

**Part II:** Leverage synthesis to build more interpretable ML models. An alternative to attention mechanism in neural networks. Applies the iterative synthesis (oracle-based) loop concepts from previous two lectures to learn ML models. The techniques here apply beyond learning from code but natural language and other data.
Probabilistic Learning from Code

Task → Probabilistic Tool → Solution

ML model

number of repositories

15 million repositories

Billions of lines of code

Google, Facebook, Microsoft

High quality, tested, maintained programs

last 5 years

github (social coding)
Why now?

Advances in Programming Languages
[Automated Reasoning, Synthesis, Constraint Solving]

Advances in Machine Learning
[Deep Learning, Graphical Models, Language Models]

Data
[> 15 million public repositories]

machine learning-based programming tools
new rules, new ideas, new opportunities

Confluence of streams
# Machine Learning for Programming

<table>
<thead>
<tr>
<th>Applications</th>
<th>Code synthesis</th>
<th>Program repair</th>
<th>Feedback generation</th>
<th>Translation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intermediate Representation</td>
<td>Sequences (sentences)</td>
<td>Translation Table</td>
<td>Trees</td>
<td>Graphical Models (CRFs)</td>
</tr>
<tr>
<td>Semantic Analysis</td>
<td>typestate analysis</td>
<td>control-flow analysis</td>
<td>alias analysis</td>
<td></td>
</tr>
<tr>
<td>Train Model (ML)</td>
<td>Neural Networks</td>
<td>SVM</td>
<td>Structured SVM</td>
<td></td>
</tr>
<tr>
<td>Query Model (ML)</td>
<td>$\text{argmax } P(y \mid x)$</td>
<td>$y \in \Omega$</td>
<td>Greedy MAP inference</td>
<td></td>
</tr>
</tbody>
</table>

More information: [http://plml.ethz.ch](http://plml.ethz.ch)

ML Systems we have built, such as [jsnice.org](http://jsnice.org) and [apk-deguard.com](http://apk-deguard.com) have > 300,000 users...
Nice2Predict.org: scalable structured prediction framework

fully, open sourced, Apache license

used by various groups worldwide

Fast, Approximate MAP inference

Fast, Parallel, Structured SVM and Pseudo-Likelihood Training

Arbitrary factors and indicator functions
### Machine Learning for Programming

#### Applications
- Code synthesis
- Deobfuscation
- Program repair
- Feedback generation
- Translation

#### Intermediate Representation
- Sequences (sentences)
- Translation Table
- Trees

#### Semantic Analysis
- Typestate analysis
- Scope analysis
- Control-flow analysis
- Alias analysis

#### Train Model (ML)
- Neural Networks
- SVM
- N-gram language model

#### Query Model (ML)
- \[ \text{argmax } P(y \mid x) \]
- \( y \in \Omega \)
- Greedy
- MAP inference

More information: [http://plml.ethz.ch](http://plml.ethz.ch)

ML Systems we have built, such as jsnice.org and apk-deguard.com have > 300,000 users...
Intuition: Scene Completion

Camera camera = Camera.open();
camera.setDisplayOrientation(90);

MediaRecorder rec = new MediaRecorder();

rec.setAudioSource(MediaRecorder.AudioSource.MIC);
rec.setVideoSource(MediaRecorder.VideoSource.DEFAULT);
rec.setOutputFormat(MediaRecorder.OutputFormat.MPEG_4);
rec.setAudioEncoder(1);
rec.setVideoEncoder(3);
rec.setOutputFile("file.mp4");
...

Statistical Code Synthesis
[Code Completion with Statistical Language Models, ACM PLDI 2014]
Camera camera = Camera.open();
camera.setDisplayOrientation(90);
camera.unlock();

MediaRecorder rec = new MediaRecorder();

rec.setCamera(camera);

rec.setAudioSource(MediaRecorder.AudioSource.MIC);
rec.setVideoSource(MediaRecorder.VideoSource.DEFAULT);
rec.setOutputFormat(MediaRecorder.OutputFormat.MPEG_4);
rec.setAudioEncoder(1);
rec.setVideoEncoder(3);

rec.setOutputFile("file.mp4");

...
Regularities in code are similar to regularities in natural language
Key Insight

Regularities in code are similar to regularities in natural language

We want to learn that

```java
    MediaRecorder rec = new MediaRecorder();
```

is before

```java
    rec.setCamera(camera);
```
Regularities in code are similar to regularities in natural language

We want to learn that
    MediaRecorder rec = new MediaRecorder();
is before
    rec.setCamera(camera);

like in natural languages
    Hello
is before
    World!
The SLANG System

Completion phase

Sentences with holes

Semantic Analysis

Query

Combine

Completed sentences

Training phase

Sentences

Semantic Analysis

Train Language Model

Language model

Sentences

Partial program

Camera camera = Camera.open();
camera.setDisplayOrientation(90);
camera.unlock();
MediaRecorder rec = new MediaRecorder();
rec.setCamera(camera);
rec.setAudioSource(MediaRecorder.AudioSource.MIC);
rec.setVideoSource(MediaRecorder.Vid eoSource.DEFAULT);
rec.setOutputFormat(MediaRecorder.O utputFormat.MPEG_4);
rec.setAudioEncoder(rec.ClockSource.MIC); rec.setOutputFile("file.mp4");
...

Synthesized program

Camera camera = Camera.open();
camera.setDisplayOrientation(90);
camera.unlock();
MediaRecorder rec = new MediaRecorder();
rec.setCamera(camera);
rec.setAudioSource(MediaRecorder.AudioSource.MIC);
rec.setVideoSource(MediaRecorder.VideoSource.DEFAULT);
rec.setOutputFormat(MediaRecorder.OutputFormat.MPEG_4);
rec.setAudioEncoder(1);
rec.setVideoEncoder(3);
rec.setOutputFile("file.mp4");
...

Semantic Analysis

Combine

Query

Completion phase

Semantic Analysis

Train Language Model

Training phase
The SLANG System

Training phase

Semantic Analysis

sentences

Language model

Train Language Model
Step1: From Programs to Sentences

she = new X();
me = new Y();
me.sleep();
if (random()) {
   me.eat();
}
she.enter();
me.talk(she);

A note on:

learning from code vs. learning from natural language (NLP):

Unlike natural language, code has formal semantics that one can benefit from if they have a way to automatically extract these...

This means that with code the actual data can be:

Choice 1  (Syntactic): a sequence of tokens (words) that make up the program. This follows the NLP mindset.

Choice 2  (Semantic): a sequence of words where each word captures something about the semantics of the program

Lets see how Choice 2 works...
From Programs to Sentences

she = new X();
me = new Y();
me.sleep();
if (random()) {
    me.eat();
}

she.enter();
me.talk(she);

We get 3 sequences:

for abstract object me:
    Y_init  sleep  talk
    Y_init  sleep  eat  talk

for abstract object she:
    X_init  enter  talk_{param1}

abstract (semantic) object:
think of it as the variable denoting all concrete objects assigned to that variable. So many objects assigned to the she variable will all be mapped to the same she abstract object. In our example we only have 1 concrete object but if we had a loop around she=new X(), we would have more concrete objects but still only 1 abstract object.
Step 2: Learn Regularities in Sentences

- Training phase
- Semantic Analysis
- Train Language Model
- Language model
- sentences
Learn Regularities

Learn regularities in obtained sentences

Regularities in sentences $\iff$ regularities in code usage

If we see many sequences like:

$$X_{\text{init}} \; \text{enter} \; \text{talk}_{\text{param1}}$$

then we should learn that $\text{talk}_{\text{param1}}$ is often after $\text{enter}$
Statistical Language Models

Given a sentence $s = w_1 \, w_2 \, w_3 \ldots \, w_n$

estimate $P( w_1 \, w_2 \, w_3 \ldots \, w_n )$

Decomposed to conditional probabilities

$P( w_1 \, w_2 \, w_3 \ldots \, w_n ) = \prod_{i=1..n} P( w_i \mid w_1 \ldots w_{i-1} )$
Statistical Language Models

Given a sentence $s = w_1 \, w_2 \, w_3 \ldots \, w_n$

estimate $P(\, w_1 \, w_2 \, w_3 \ldots \, w_n \,)$

Decomposed to conditional probabilities (via chain rule):

$$P(\, w_1 \, w_2 \, w_3 \ldots \, w_n \,) = \prod_{i=1..n} P(\, w_i \mid w_1 \ldots w_{i-1} \,)$$

$P(\, \text{The quick brown fox jumped} \,) =$

$P(\, \text{The} \,)$ $P(\, \text{quick} \mid \text{The} \,)$ $P(\, \text{brown} \mid \text{The quick} \,)$ $P(\, \text{fox} \mid \text{The quick brown} \,)$ $P(\, \text{jumped} \mid \text{The quick brown fox} \,)$
Choice I: N-gram language model

Conditional probability based only on previous $n-1$ words

$$P(w_i \mid w_1 \ldots w_{i-1}) \approx P(w_i \mid w_{i-n+1} \ldots w_{i-1})$$
N-gram language model

Conditional probability based only on previous n-1 words

\[ P( w_i \mid w_1 \ldots w_{i-1} ) \approx P( w_i \mid w_{i-n+1} \ldots w_{i-1} ) \]

n-1 words
N-gram language model

Conditional probability based only on previous $n-1$ words

\[
P(w_i \mid w_1 \ldots w_{i-1}) \approx P(w_i \mid w_{i-n+1} \ldots w_{i-1})
\]

Training is achieved by counting $n$-grams, called MLE = maximum likelihood estimation. E.g., with 3-gram language model, we get:

\[
P(\text{jumped} \mid \text{The quick brown fox}) \approx P(\text{jumped} \mid \text{brown fox}) \approx \frac{\#(\text{brown fox jumped})}{\#(\text{brown fox})}
\]

#(n-gram) - number of occurrences of $n$-gram in training data

Time complexity for each word encountered in training is constant, so training is usually fast.

* technicality: to actually get a probability distribution across all sentences we typically add N-1 start symbols and N-1 end symbols </s> to each sentence.
3-gram language model: example

\[ P(w_1 \cdot w_2 \cdot w_3 \cdot \ldots \cdot w_n) \approx P(w_1) \cdot P(w_2 | w_1) \cdot P(w_3 | w_1 \cdot w_2) \cdot \ldots \cdot P(w_n | w_{n-2} \cdot w_{n-1}) \]

<table>
<thead>
<tr>
<th>3-grams</th>
<th># of occurrences</th>
</tr>
</thead>
<tbody>
<tr>
<td>brown fox jumped</td>
<td>125</td>
</tr>
<tr>
<td>brown fox walked</td>
<td>45</td>
</tr>
<tr>
<td>brown fox snapped</td>
<td>30</td>
</tr>
</tbody>
</table>

\[ P(\text{jumped} | \text{brown fox}) \approx \frac{125}{200} \approx 0.625 \]

\[ P(\text{brown fox jumped}) \approx \frac{P(\text{brown}) \cdot P(\text{fox} | \text{brown}) \cdot P(\text{jumped} | \text{brown fox})}{P(\text{brown}) \cdot P(\text{fox} | \text{brown}) \cdot P(\text{jumped} | \text{brown fox})} \approx \frac{200 / 600 \cdot 200 / 200 \cdot 125 / 200}{200 / 600 \cdot 200 / 200 \cdot 125 / 200} \approx 0.208 \]
Key Problem: Sparsity of Data

What if this number is 0?

\[
P( \text{jumped} | \text{brown fox} ) \approx \frac{\#( \text{brown fox jumped} )}{\#( \text{brown fox} )}
\]

The problem of sparsity gets worse as the size of the n-gram becomes larger.

We need to handle n-grams with 0 or few occurrences in the training data. Techniques that can do that are: smoothing, discounting
Solution: Smoothing Techniques

- Smoothing techniques give non-zero probability to n-grams not in the training data.

- Essentially, they try to estimate how likely it is that the n-gram is missing due to the limited size of the training data.

- They work by taking the probability mass of the existing n-grams and redistributing that mass over n-grams that occur zero times.

- Typically, probability of such n-grams is estimated by looking at the probabilities of n-1 grams, n-2 grams, unigrams, etc.

Example smoothing techniques are: Witten-Bell Interpolated (WBI), Witten-Bell Backoff (WBB), Natural Discounting (ND), Stupid-Backoff (SB)
Smoothing: Intuitively

Distribution of probability mass before smoothing

Training Data

Smoothing

Distribution of probability mass after smoothing

Training Data

All other sentences get 0 probability

All other sentences get non-zero probability
N-gram language model

Conditional probability only on previous $n-1$ words

$$P(w_i \mid w_1 \ldots w_{i-1}) \approx P(w_i \mid w_{i-n+1} \ldots w_{i-1})$$

Existing library implementing language models is SRILM:
http://www.speech.sri.com/projects/srilm/
Choice II: Recurrent Neural Networks (RNN)

- **Output vector:** $o_0$, $o_1$, $o_2$
  - Each entry is a probability of a word.

- **Hidden layer:**
  - $s_0$, $s_1$, $s_2$
  - Captures "history".

- **Input vector:**
  - $x_0$, $x_1$, $x_2$
  - Open, Write, Read
  - A 1 denotes an input word.

Matrices $U$, $V$, $W$ learned through backprop.

Matrix $U$ is a low-dimensional continuous vector representation of input words, also referred to as "word embedding."
RNNs: more details on board
Choice II: Recurrent Neural Networks (RNN)

+ can learn dependencies beyond the prior several words (e.g., LSTM)
+ learns continuous representation of vocabulary (helps avoid sparsity)
- slow and tricky to train
- predictions hard to explain


TensorFlow
[https://www.tensorflow.org/](https://www.tensorflow.org/)
ML-based Code Synthesis

```java
smsMgr = SmsManager.getDefault();
int length = message.length();
if (length > MAX_SMS_MESSAGE_LENGTH) {
    list = smsMgr.divideMessage(message);
    ? {smsMgr, list} // (Hole H1)
} else {
    ? {smsMgr, message} // (Hole H2)
}
```
smsMgr = SmsManager.getDefault();
int length = message.length();
if (length > MAX_SMS_MESSAGE_LENGTH) {
    list = smsMgr.divideMessage(message);
    ? {smsMgr, list} // (Hole H1)
} else {
    ? {smsMgr, message} // (Hole H2)
}
ML-based Code Synthesis

get\textsubscript{Default} \_\text{result} \quad \text{divideMessage} \quad \text{H1}

get\textsubscript{Default} \_\text{result} \quad \text{H2}

\text{divideMessage} \_\text{result} \quad \text{H1}

\text{length} \quad \text{H2}
<table>
<thead>
<tr>
<th>Code Pattern</th>
<th>Completion Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>getDefault</strong> result <strong>divideMessage</strong> <strong>sendMultipartTextMessage</strong></td>
<td>0.0033</td>
</tr>
<tr>
<td><strong>getDefault</strong> result <strong>divideMessage</strong> <strong>sendTextMessage</strong></td>
<td>0.0016</td>
</tr>
<tr>
<td><strong>getDefault</strong> result <strong>sendTextMessage</strong></td>
<td>0.0073</td>
</tr>
<tr>
<td><strong>getDefault</strong> result <strong>sendMultipartTextMessage</strong></td>
<td>0.0010</td>
</tr>
<tr>
<td><strong>divideMessage</strong> result <strong>sendMultipartTextMessage</strong></td>
<td>0.0821</td>
</tr>
<tr>
<td><strong>length</strong> <strong>length</strong></td>
<td>0.0132</td>
</tr>
<tr>
<td><strong>length</strong> <strong>split</strong></td>
<td>0.0080</td>
</tr>
<tr>
<td><strong>length</strong> <strong>sendTextMessage</strong> <strong>param3</strong></td>
<td>0.0017</td>
</tr>
</tbody>
</table>

**ML-based Code Synthesis**
# ML-based Code Synthesis

<table>
<thead>
<tr>
<th>Function Call</th>
<th>Completion Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getDefault(result)</code></td>
<td>0.0033</td>
</tr>
<tr>
<td><code>divideMessage sendMultipartTextMessage</code></td>
<td>0.0016</td>
</tr>
<tr>
<td><code>sendTextMessage</code></td>
<td>0.0073</td>
</tr>
<tr>
<td><code>sendMultipartTextMessage</code></td>
<td>0.0010</td>
</tr>
<tr>
<td><code>divideMessage param3</code></td>
<td>0.0821</td>
</tr>
<tr>
<td><code>length</code></td>
<td>0.0132</td>
</tr>
<tr>
<td><code>length split</code></td>
<td>0.0080</td>
</tr>
<tr>
<td><code>sendTextMessage param3</code></td>
<td>0.0017</td>
</tr>
</tbody>
</table>
ML-based Code Synthesis

<table>
<thead>
<tr>
<th>Method Selection</th>
<th>Completion Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getDefault</code> result</td>
<td>0.0033</td>
</tr>
<tr>
<td><code>divideMessage</code></td>
<td>0.0016</td>
</tr>
<tr>
<td><code>sendMultipartTextMessage</code></td>
<td>0.0821</td>
</tr>
<tr>
<td><code>sendTextMessage</code></td>
<td>0.0073</td>
</tr>
<tr>
<td><code>divideMessage</code> result</td>
<td>0.0010</td>
</tr>
<tr>
<td><code>sendMultipartTextMessage</code></td>
<td>0.0017</td>
</tr>
<tr>
<td><code>length</code></td>
<td>0.0080</td>
</tr>
<tr>
<td><code>split</code></td>
<td></td>
</tr>
<tr>
<td><code>sendTextMessage</code> param3</td>
<td>0.0017</td>
</tr>
</tbody>
</table>

Not a feasible solution: completions disagree on selected method

The solution must satisfy program constraints
# ML-based Code Synthesis

<table>
<thead>
<tr>
<th>Code Sequence</th>
<th>Completion Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getDefault result divideMessage sendMultipartTextMessage</code></td>
<td>0.0033</td>
</tr>
<tr>
<td><code>getDefault result divideMessage sendTextMessage</code></td>
<td>0.0016</td>
</tr>
<tr>
<td><code>getDefault result sendTextMessage</code></td>
<td>0.0073</td>
</tr>
<tr>
<td><code>getDefault result sendMultipartTextMessage</code></td>
<td>0.0010</td>
</tr>
<tr>
<td><code>divideMessage result sendMultipartTextMessage_{param3}</code></td>
<td>0.0821</td>
</tr>
<tr>
<td><code>length length</code></td>
<td>0.0132</td>
</tr>
<tr>
<td><code>length split</code></td>
<td>0.0080</td>
</tr>
<tr>
<td><code>length sendTextMessage_{param3}</code></td>
<td>0.0017</td>
</tr>
</tbody>
</table>
## ML-based Code Synthesis

<table>
<thead>
<tr>
<th>Function Call</th>
<th>Completion Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getDefault</code></td>
<td>0.0033</td>
</tr>
<tr>
<td><code>divideMessage</code></td>
<td>0.0016</td>
</tr>
<tr>
<td><code>sendTextMessage</code></td>
<td>0.0073</td>
</tr>
<tr>
<td><code>sendMultipartTextMessage</code></td>
<td>0.0010</td>
</tr>
<tr>
<td><code>divideMessage</code></td>
<td>0.0821</td>
</tr>
<tr>
<td><code>sendMultipartTextMessage</code></td>
<td>0.0017</td>
</tr>
<tr>
<td><code>length</code></td>
<td>0.0132</td>
</tr>
<tr>
<td><code>length</code></td>
<td>0.0080</td>
</tr>
<tr>
<td><code>sendTextMessage</code></td>
<td>0.0017</td>
</tr>
</tbody>
</table>
smsMgr = SmsManager.getDefault();
int length = message.length();
if (length > MAX_SMS_MESSAGE_LENGTH) {
    list = smsMgr.divideMessage(message);
    smsMgr.sendMultipartTextMessage(...list...);
} else {
    smsMgr.sendTextMessage(...message...);
}
The SLANG System

Completion phase

Sentences with holes → Semantic Analysis → Completed sentences → Combine

Training phase

Sentences → Semantic Analysis → Train Language Model

84 testing samples

Language model

~100MB

Correct completion in top 3 for ~90% of cases

1M Java methods

~700MB

Synthesized program

Camera camera = Camera.open();
camera.setDisplayOrientation(90);
camera.unlock();
MediaRecorder rec = new MediaRecorder();
rec.setCamera(camera);
rec.setAudioSource(MediaRecorder.AudioSource.MIC);
rec.setVideoSource(MediaRecorder.VideoSource.DEFAULT);
rec.setOutputFormat(MediaRecorder.OutputFormat.MPEG_4);
rec.setAudioEncoder(1);
rec.setVideoEncoder(3);
rec.setOutputFile("file.mp4");

Partial program

? Camera camera = Camera.open();
camera.setDisplayOrientation(90);
camera.unlock();
MediaRecorder rec = new MediaRecorder();
rec.setCamera(camera);
rec.setAudioSource(MediaRecorder.AudioSource.MIC);
rec.setVideoSource(MediaRecorder.VideoSource.DEFAULT);
rec.setOutputFormat(MediaRecorder.OutputFormat.MPEG_4);
rec.setAudioEncoder(1);
rec.setVideoEncoder(3);
rec.setOutputFile("file.mp4");

Semantic
Analysis

Query

Combine

84 testing samples

Semantic
Analysis

Train
Language
Model
Do semantics help?

**precision vs. % of data used**

- Choice 1: treat code as tokens (NLP way)
- Choice 2: treat code as semantic entity

*need to decide what to `compile` into a word

**Semantic analysis benefit = 10x more data**
Next: Synthesis for Interpretable ML
Fundamental Question

Data → Learning → Model

- Widely Applicable
- Efficient Learning
- High Precision
- Interpretable Model

Probabilistic Model