Machine Programming

Justin Gottschlich, Intel Labs

December 12th, 2018

TVM Conference, University of Washington
Motivation

*We have a software programmer resource problem*
Motivation

We have a software programmer resource problem
Motivation

*We have a software programmer resource problem*

2019 human population \(*7,714M*\)

2019 developers \(*26.4M*\)

% of programmers: \(*>0.34\%*\)
Motivation

We have a software programmer resource problem

2019 human population 7,714M
2019 developers 26.4M
% of programmers: > 0.34% <

2019 human population 7,714M
2019 developers 1,200M
% of drivers: > 15.56% <
Motivation

2019 human population 7,714M
2019 developers 26.4M
% of programmers: > 0.34% <

2019 human population 7,714M
2019 developers 1,200M
% of drivers: > 15.56% <
Motivation

What if programming could be as simple as driving?

How can we simplify programming (mostly with machine learning)?

(1) Reduce intention-challenge, (2) delegate most work to machines.
Human programming vs machine programming
Human Programming

The process of developing software, principally by one or more humans.

- **Examples**
  - Writing code in *<your favorite language here>*

- **Pros**
  - Near complete control over the software created, exact behaviors

- **Cons**
  - Expensive, slow, error-prone, human-resource limited
Machine Programming

The process of developing software where some or all of the steps are performed autonomously.

- **Examples**
  - *Classical*: compiler transformations
  - *Emerging*: Verified lifting[1], AutoTVM[2], Sketch[3], DeepCoder[4], SapFix/Sapienz[5]

- **Pros**
  - Resource constrained by computers, most humans can create software

- **Cons**
  - Immature, may lack full control, may be partially stochastic
The Three Pillars of Machine Programming (MP)
MAPL/PLDI’18

Justin Gottschlich, Intel
Armando Solar-Lezama, MIT
Nesime Tatbul, Intel
Michael Carbin, MIT
Martin, Rinard, MIT
Regina Barzilay, MIT
Saman Amarasinghe, MIT
Joshua B Tenenbaum, MIT
Tim Mattson, Intel
Examples of the Three Pillars of MP

- **Intention**
  - “Automating String Processing in Spreadsheets using Input-Output Examples” (Sumit Gulwani)
  - “Program Synthesis by Sketching” (Armando Solar-Lezama, Adviser: R. Bodik)

- **Invention**
  - “The Case for Learned Index Structures” (Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, Neoklis Polyzotis)

- **Adaptation**
  - “Precision and Recall for Time Series” (Nesime Tatbul, TJ Lee, Stan Zdonik, Mejbah Alam, Justin Gottschlich)

**Adaptation**

Anomaly Detection Interpretability
(Xin Sheng, Mejbah Alam, Justin Gottschlich, Armando Solar-Lezama)
Flash Fill

- **Intention**
  - “Automating String Processing in Spreadsheets using Input-Output Examples” (Sumit Gulwani)
  - “Program Synthesis by Sketching” (Armando Solar-Lezama, Adviser: R. Bodik)

- **Invention**
  - “The Case for Learned Index Structures” (Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, Neoklis Polyzotis)

- **Adaptation**
  - “Precision and Recall for Time Series” (Nesime Tatbul, TJ Lee, Stan Zdonik, Mejbah Alam, Justin Gottschlich)
Sketch

- **Intention**
  - “Automating String Processing in Spreadsheets using Input-Output Examples” (Sumit Gulwani)
  - “Program Synthesis by Sketching” (Armando Solar-Lezama, Adviser: R. Bodik)

- **Invention**
  - “The Case for Learned Index Structures” (Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, Neoklis Polyzotis)

- **Adaptation**
  - “Precision and Recall for Time Series” (Nesime Tatbul, TJ Lee, Stan Zdonik, Mejbah Alam, Justin Gottschlich)
Learned Index Structures

- **Intention**
  - “Automating String Processing in Spreadsheets using Input-Output Examples” (Sumit Gulwani)
  - “Program Synthesis by Sketching” (Armando Solar-Lezama, Adviser: R. Bodik)

- **Invention**
  - “The Case for Learned Index Structures” (Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, Neoklis Polyzotis)

- **Adaptation**
  - “Precision and Recall for Time Series” (Nesime Tatbul, TJ Lee, Stan Zdonik, Mejbah Alam, Justin Gottschlich)

Time Series Anomalies and Interpretability

- **Intention**
  - “Automating String Processing in Spreadsheets using Input-Output Examples” (Sumit Gulwani)
  - “Program Synthesis by Sketching” (Armando Solar-Lezama, Adviser: R. Bodik)

- **Invention**
  - “The Case for Learned Index Structures” (Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, Neoklis Polyzotis)

- **Adaptation**
  - “Precision and Recall for Time Series” (Nesime Tatbul, TJ Lee, Stan Zdonik, Mejbah Alam, Justin Gottschlich)

---

**Range-based Anomalies**

```
<table>
<thead>
<tr>
<th>Real</th>
<th>Predicted</th>
</tr>
</thead>
<tbody>
<tr>
<td>R₁</td>
<td>P₁</td>
</tr>
<tr>
<td>R₂</td>
<td>P₂</td>
</tr>
<tr>
<td>R₃</td>
<td></td>
</tr>
</tbody>
</table>
```

- **Adaptation**
  - Anomaly Detection Interpretability
    (Xin Sheng, Mejbah Alam, Justin Gottschlich, Armando Solar-Lezama)

---

Adaptation

Software that automatically evolves (e.g., repairs, optimizes, secures) itself

Adaptation is principally about range-based anomaly detection
Time Series Anomaly Detection

Point-based Anomalies

- False Negatives (FN)
- True Positives (TP)
- False Positives (FP)

Range-based Anomalies

- Partial overlap at back-end of R₁ and front-end of P₁
- Two real anomaly ranges R₂ and R₃ overlapping with one predicted anomaly range P₂

- How do we define TPs, TNs, FPs, FNs?
(Prior) State of the Art

- Classical recall/precision
  - **Point-based anomalies**
  - Recall penalizes FN, precision penalizes FP
  - $F_\beta$-measure to combine & weight them

- Numenta Anomaly Benchmark (NAB)'s Scoring Model [1]
  - **Point-based anomalies**
  - Focuses specifically on early detection use cases
  - Difficult to use in practice (irregularities, ambiguities, magic numbers) [2]

- Activity recognition metrics
  - No support for flexible time bias

\[ F_\beta = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{\beta^2 \cdot \text{precision} + \text{recall}} \]

$\beta$ : relative importance of Recall to Precision
$\beta = 1$ : evenly weighted (harmonic mean)
$\beta = 2$ : weights Recall higher (i.e., no FN!)
$\beta = 0.5$ : weights Precision higher (i.e., no FP!)

(Prior) State of the Art

- Classical recall/precision
  - **Point-based anomalies**
  - Recall penalizes FN, precision penalizes FP
  - $F_\beta$-measure to combine & weight them

- Numenta Anomaly Benchmark (NAB)'s Scoring Model [1]
  - **Point-based anomalies**
  - Focuses specifically on early detection use cases
  - Difficult to use in practice (irregularities, ambiguities, magic numbers) [2]

- Activity recognition metrics
  - No support for flexible time bias

\[
F_\beta = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{\beta^2 \cdot \text{precision} + \text{recall}}
\]

$\beta$ : relative importance of Recall to Precision
$\beta = 1$ : evenly weighted (harmonic mean)
$\beta = 2$ : weights Recall higher (i.e., no FN!)
$\beta = 0.5$ : weights Precision higher (i.e., no FP!)

New Evaluation Model

Expressive, Flexible, Extensible

- Superset of:
  - Classical model
  - Other state-of-the-art evaluators (NAB)
- NeurIPS ‘18 Spotlight

- Key: evaluate anomaly detectors with practical meaningfulness

Precision & Recall for Time Series

<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R, R_i^j$</td>
<td>set of real anomaly ranges, the $i^{th}$ real anomaly range</td>
</tr>
<tr>
<td>$P, P_j^k$</td>
<td>set of predicted anomaly ranges, the $j^{th}$ predicted anomaly range</td>
</tr>
<tr>
<td>$N, N_r, N_p$</td>
<td>number of all points, number of real anomaly ranges, number of predicted anomaly ranges</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>relative weight of existence reward</td>
</tr>
<tr>
<td>$\gamma()$, $\omega()$, $\delta()$</td>
<td>overlap cardinality function, overlap size function, positional bias function</td>
</tr>
</tbody>
</table>

Range-based Recall:

\[
\text{Recall}_T(R, P) = \frac{\sum_{i=1}^{N_r} \text{Recall}_T(R_i^j, P)}{N_p} \\
\text{Recall}_T(R_i^j, P) = \alpha \times \text{Existence Reward}(R_i^j, P) + (1 - \alpha) \times \text{Overlap Reward}(R_i^j, P) \\
\text{Existence Reward}(R_i^j, P) = \begin{cases} 1, & \text{if } \sum_{j=1}^{N_p} |R_i^j \cap P_j| \geq 1 \\ 0, & \text{otherwise} \end{cases} \\
\text{Overlap Reward}(R_i^j, P) = \text{Cardinality Factor}(R_i^j, P) \times \sum_{j=1}^{N_p} \omega(R_i^j, R_i^j \cap P_j, \delta) \\
\text{Cardinality Factor}(R_i^j, P) = \begin{cases} 1, & \text{if } R_i^j \text{ overlaps with at most one } P_j \subseteq P \\ \gamma(R_i^j, P), & \text{otherwise} \end{cases}
\]

Range-based Precision:

\[
\text{Precision}_T(R, P) = \frac{\sum_{i=1}^{N_r} \text{Precision}_T(R_i^j, P)}{N_p} \\
\text{Precision}_T(R_i^j, P) = \text{Cardinality Factor}(P_i^k, R) \times \sum_{j=1}^{N_p} \omega(P_i^k, P_i^k \cap R_j, \delta)
\]
TSAD-Evaluator Overview

- A tool that implements our customizable evaluation model

- Can be used in two modes:
  -c: compute classical metrics (point-based)
  -t: compute time series metrics (range-based)

- Input:
  2 files with anomaly labels (e.g., simple.real, simple.pred)
  Evaluator parameters

- Output:
  Precision, Recall, F-Score

- A library of pre-defined choices for $\gamma()$ and $\delta()$
  + templates for user-defined extensions

- Example:

  ```
  ./evaluate -t simple.real simple.pred 1 0 reciprocal flat front
  ```

https://github.com/IntelLabs/TSAD-Evaluator/
New Evaluation Model – Helps Intel

- Positioned to benefit Intel internally
  - Cyber-security, data centers, **SW/HW vulnerabilities**
Anomaly Detection Interpretability

Analysis of a anomaly:

1. **Where/when** is the anomaly?
   - Existing work can achieve this

2. **Why** is this an anomaly?
   - Partial solutions in this space

3. **How to fix the anomaly**?
   - Mostly an open problem

The “**How**” and “**Why**” are open questions for anomaly detection & neural networks
AutoPerf: ZPL using Autoencoder

Used to detect parallel software performance anomalies

- Encodes input data to a reduced dimension (encoder)
- Reconstructs input data as target of the network (decoder)
- Reconstruction error:

Anomalous data cannot be reconstructed using representation learned from non-anomalous data

Train autoencoder using non-anomalous dataset → Determine a threshold for anomaly using reconstruction error → Inferencing for new inputs → Reconstruction error > threshold?

Yes → Anomaly detected

No → Non-anomalous
Interpreting Neural Network Judgments

Polaris: Corrections as Explanations for Neural Network Judgment. [2]

- Judgment problem: binary classification problem where one output is preferred
  - Vehicle collision, software performance and correct bugs, security vulnerabilities
- Proposed solution: corrections as actionable explanations.

- Desired properties:
  - Minimal
  - Stable
  - Symbolic

[2] Interpreting Neural Network Judgments via Minimal, Stable, and Symbolic Corrections, Xin Zhang (MIT), Armando Solar-Lezama (MIT), Rishabh Singh (Google Brain), [NIPS ’18 (to appear)]
IL+MIT: Interpreting AutoPerf using Polaris

**Goal:** automatic identification & correction of adaptation-like anomalies
Early Results: Anomaly Detection Interpretability
Early Results:
Anomaly Detection
Interpretability
Early Results:
Anomaly Detection
Interpretability

[Graph showing a scatter plot with labeled areas for non-anomalous space and anomaly, along with a point labeled AutoPerf.]
Early Results: Anomaly Detection Interpretability

non-anomalous space

anomaly

Polaris

AutoPerf
Early Results: Anomaly Detection Interpretability

Action: move to non-anomalous space by reducing L3 HITMs
Learning to Optimize Tensor Programs

Figure 2: Framework for learning to optimize tensor programs.
AutoTVM (NeurIPS '18 Spotlight)

Learning to Optimize Tensor Programs

Tianqi Chen¹  Lianmin Zheng²  Eddie Yan¹  Ziheng Jiang¹  Thierry Moreau¹  Luis Ceze¹  Carlos Guestrin¹  Arvind Krishnamurthy¹
¹Paul G. Allen School of Computer Science & Engineering, University of Washington
²Shanghai Jiao Tong University

Figure 11: End-to-end performance across back-ends. *AutoTVM outperforms the baseline methods.

Learning to Optimize Tensor Programs

Figure 11: End-to-end performance across back-ends. AutoTVM outperforms the baseline methods.
Higher is faster
Performance Results for MB=28, 85% of LIBXSMM

Higher is faster

Research by Intel Labs (Anand Venkat, Michael Anderson, Alexander Heinecke, Evangelos Georganas)
Conclusion

- **Machine programming is coming!**
  - Interested in collaborating? Please contact me!
  - Teaching machine programming course @ Penn (Spring 2020)

- **Machine Learning and Programming Languages (MAPL) Workshop**
  - Please consider submitting a paper to MAPL ‘19 (@ PLDI ‘19)
    - 10 page ACM SIGPLAN published proceedings (submission: Jan/Feb)
  - General Chair: Tim Mattson (Intel Labs)
  - Program Chair: Armando Solar-Lezama (MIT)
Questions?

justin.gottschlich@intel.com