Analyzing Software using Deep Learning

Introduction

Subscribe to the course via Piazza:
piazza.com/tu-darmstadt.de/summer2018/20000999iv

Prof. Dr. Michael Pradel
Software Lab, TU Darmstadt
About Me

- Michael Pradel
- At TU Darmstadt since 2014

Before joining TUDA
- Master-level studies in Dresden and Paris
- Master thesis at EPFL, Switzerland
- PhD at ETH Zurich, Switzerland
- Postdoctoral researcher at UC Berkeley, USA
About the Software Lab

- My research group since 2014
- Focus: Tools and techniques for building reliable, efficient, and secure software
  - Program analysis
  - Test generation
- Thesis and job opportunities
Plan for Today

■ Introduction
  □ What the course is about
  □ Why it is interesting
  □ How it can help you

■ Organization
  □ Lectures and final exam
  □ Course project

■ Basics
  □ Program analysis
  □ Deep learning
What is Program Analysis?

- Automated analysis of program behavior, e.g., to
  - find programming errors
  - optimize performance
  - find security vulnerabilities

Input → Program → Output
What is Program Analysis?

- Automated analysis of program behavior, e.g., to
  - find programming errors
  - optimize performance
  - find security vulnerabilities
What is Program Analysis?

- Automated analysis of **program behavior**, e.g., to
  - find programming errors
  - optimize performance
  - find security vulnerabilities

![Diagram](image-url)
Why Do We Need It?

Basis for various tools that make developers productive

- Compilers
- Bug finding tools
- Performance profilers
- Code completion
- Automated testing
- Code summarization/documentation
Traditional Approaches

- Analysis has **built-in knowledge** about the problem to solve
- Significant human effort to create a program analysis
  - Conceptual challenges
  - Implementation effort
- Analyze a **single program** at a time
Learning from Existing Data

- Huge amount of existing code ("big code")
- Programs are regular and repetitive
- Machine learning: Extract knowledge and apply in new contexts
- Learn how to ..
  - .. complete partial code
  - .. use an API
  - .. fix programming errors
  - .. create inputs for testing
Deep Learning

Class of machine learning algorithms

- Neural network architectures
- "Deep" = multiple layers
- Features and representation of inputs are extracted automatically

Revolutionizes entire areas
This Course

Intersection of program analysis and deep learning

- Some of the basics:
  E.g., program representations, neural network architectures

- State of the art research results:
  Based on recent research papers

- Hands-on experience:
  Coding project
Not This Course

What this course is not about

- Detailed coverage of program analysis
- Detailed coverage of machine learning
- Programming tutorial for TensorFlow

Check out related courses

- E.g., ”Program Testing and Analysis” (winter semester)
Plan for Today

- **Introduction**
  - What the course is about
  - Why it is interesting
  - How it can help you

- **Organization**
  - Lectures and final exam
  - Course project

- **Basics**
  - Program analysis
  - Deep learning
Organization

■ Weekly meetings
  □ 6 lectures
  □ 6 Q&A sessions for course project

■ Reading material

■ 2nd half of semester (from May 14):
  □ Course project

■ July 9: Submission of project

■ Exam period: Written exam
Grading

50% written exam
- Content of lectures and reading material
- Open book, one hour
- Will test your understanding, not your memory

50% course project
- Effectiveness of your implementation
- Documentation and code quality
Piazza

Platform for discussions, in-class quizzes, and sharing additional material

- Please register and enroll for the class
- Use it for all questions related to the course
- Messages sent to all students go via Piazza (not TUCaN!)

Subscribe to the course via Piazza:
piazza.com/tu-darmstadt.de/summer2018/20000999iv
Learning Material

There is no script or single book that covers everything

- Slides and hand-written nodes: Available after lecture
- Pointers to papers, book chapters, and web resources
Course Project

- Individual project
- Same task for everybody
- Implement and evaluate a neural network that predicts/generates code
- Based on existing tools
  - TensorFlow library for machine learning
  - Python
- More details on May 14
Plan for Today

■ Introduction
  □ What the course is about
  □ Why it is interesting
  □ How it can help you

■ Organization
  □ Lectures and final exam
  □ Course project

■ Basics
  □ Program analysis
  □ Deep learning
Program Analysis

Many ways to represent (parts of) a program

- Sequence of characters
- Sequence of tokens
- Abstract syntax tree
- Control flow graph
- Call graph
- etc.
Program Analysis

Many ways to represent (parts of) a program

- Sequence of characters
- **Sequence of tokens**
- Abstract syntax tree
- Control flow graph
- Call graph
- etc.
Tokens

Tokenizer (or lexer)
- Part of compiler
- Splits sequence of characters into subsequences called tokens

E.g., for Java, six kinds of tokens:
- Identifiers, e.g., MyClass
- Keywords, e.g., if
- Separators, e.g., . or { 
- Operators, e.g., * or ++
- Literals, e.g., 23 or "hi"
- Comments, e.g., /* bla */
Tokens: Example

```python
if (flag == true) {
    name = "joe";
}
```
Abstract Syntax Tree

- **Tree** representation of source code
- ”Abstract” because some details of syntax omitted
  - E.g., `{` in Java
- **Nodes**: Construct in source code
- **Edges**: Parent-child relationship
- Check out **Esprima** for obtaining ASTs of Javascript:
  - http://esprima.org/demo
Abstract Syntax Trees: Example

Example: JavaScript

```
var x = 6 * y;
```
Deep Learning: Example

Example: **Handwriting recognition**

- **Goal:** Recognize digits 0..9
- **Easy for a human but challenging for a computer**
- **Idea:** Learn from a large number of *training examples*
- **Deep learning:** > 99% accuracy

Following slides based on Chapter 1 of neuralnetworksanddeeplearning.com
Network of neurons

E.g. pixels of an image

E.g., whether it's the digit 3
Perceptions

Most basic kind of neuron

- binary inputs
- binary output

\[
\begin{align*}
\text{output} &= \begin{cases} 
0 & \text{if } \sum w_j x_j \leq \text{threshold} \\
1 & \text{if } \sum w_j x_j > \text{threshold}
\end{cases} \\
&= \begin{cases} 
0 & \text{if } w \cdot x + b \leq 0 \\
1 & \text{if } w \cdot x + b > 0
\end{cases}
\end{align*}
\]

\(w\) ... weights
\(b\) ... bias
Example

$x_1 = \text{Weather is good}$

$x_2 = \text{Friends go}$

$x_3 = \text{Like cheese}$

$\omega_0 = 5$

$\omega_2 = 3$

$\omega_3 = 7$

Bias: $-9$

Output: $\text{Go to cheese festival}$

Assume: $x_1 = 1, \ x_2 = 1, \ x_3 = 0$

Quiz: Output? $w \cdot x = 5 \cdot 1 + 3 \cdot 1 + 0 \cdot 1 = 8$

Output =

\[
\begin{cases} 
0 & \text{if } 8 - 7 \leq 0 \\
1 & \text{if } 8 - 7 > 0 \rightarrow \text{Go to festival} 
\end{cases}
\]
### Computing Logical Functions

#### NAND gate

- \( x_1 \) with \( w_1 = -2 \) and bias = 3
- \( x_2 \) with \( w_2 = -2 \)

<table>
<thead>
<tr>
<th>( x_1 )</th>
<th>( x_2 )</th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Universal Computation

- Networks of NAND perceptrons can simulate every circuit containing only NAND gates
- Can express arbitrary computations!
Example: Adding Two Bits

NAND gate:

\[ x_1 \]
\[ x_2 \]

\[ \text{sum: } x_1 \oplus x_2 \]
\[ \text{carry bit: } x_1 x_2 \]

Network of perceptrons:
Challenge: Set Weights and Biases

- More complex networks can perform arbitrary computations

- How to decide on the weights and biases?

  - Option 1: Hand-tune them
    → Infeasible for complex networks

  - Option 2: Learn them
    → Key idea behind machine learning with neural networks
Making Learning Possible

Want: Small change of weights & biases causes small change of output

Problem: Perception doesn't have this property

output = \text{step}(w \cdot x + b)
Sigmoid Neuron

\[ \text{output} = \sigma \left( w \cdot x + b \right) \]

sigmoid function:

\[ \sigma(z) = \frac{1}{1 + e^{-z}} = \frac{1}{1 + \exp(-\sum j w_j x_j + b)} \]

- Enables learning:
  - Small change causes small change
Activation Functions

**Step function**

**Sigmoid/logistic function**

**Identity function**

**Rectified linear unit**

Different activation functions are useful in different kinds of networks.
Feedforward Network

- Output of layer n-1 is input to layer n
- No loops; information is never fed back
- Useful if input/output pairs are independent of each other
  - E.g., recognize digit
Recurrent Neural Networks

- Output of layer $n$ may be fed back to layers $n, n-1, \ldots$
- Back edges serve as "memory"
- Useful for sequential information
  - Input/output pair depend on each other
**Learning: Cost Function**

- Cost function = feedback on how good the output is for a given input

- Example:
  - Network:
    - prob(0)
    - prob(1)
    - prob(9)

  Length of vector:
  \[
  \| (x, y, z) \| = \sqrt{x^2 + y^2 + z^2}
  \]

  \[
  C(w, b) = \frac{1}{2 \cdot n} \sum_{x} \| y(x) - a \| ^2
  \]

  nb. of training inputs  desired  output of network

  If digit is known to be 6, want output:
  \[y(x) = (0, 0, 0, 0, 0, 0, 1, 0, 0, 0)^T\]
  
  Actual output may be:
  \[a = (0, 0, 0, 0, 2, 0, 0, 0.7, 0.1, 0, 0)^T\]
  
  Quadratic cost function (or mean squared error)
**Quiz: Cost Function**

- Recognition of hand-written digits
- Only digits 0, 1, and 2
- Training examples:

<table>
<thead>
<tr>
<th>Example</th>
<th>Actual</th>
<th>Desired</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>((0, 1, 0)^T)</td>
<td>((0.5, 0.5, 0)^T)</td>
</tr>
<tr>
<td>2</td>
<td>((1, 0, 0)^T)</td>
<td>((1, 0, 0)^T)</td>
</tr>
</tbody>
</table>

- What is the value of the cost function?
Unit: Cost Function

\[ C(w, b) = \frac{1}{2 \cdot n} \sum_x \| y(x) - a \|^2 \]

\[ = \frac{1}{2 \cdot 2} \cdot (\| (-0.5, 0.5, 0)^T \|^2 + \| (0, 0, 0)^T \|^2) \]

\[ = \frac{1}{4} \cdot (0.5 + 0) = 0.125 \]
Goal: Minimize Cost Function

- **Goal of learning**: Find weights and biases that **minimize the cost function**

- **Approach**: **Gradient descent**
  - Compute **gradient** of $C$: Vector of partial derivatives
  - "Move" closer toward minimum step-by-step
  - **Learning rate** determines step size
Training Examples

- Effort of computing gradient depends on number of examples

- **Stochastic gradient descent**
  - Use small sample of all examples
  - Compute estimate of true gradient

- **Epochs and mini-batches**
  - Split training examples into \( k \) mini-batches
  - Train network with each mini-batch
  - Epoch: Each mini-batch used exactly once
Plan for Today

■ Introduction
  □ What the course is about
  □ Why it is interesting
  □ How it can help you

■ Organization
  □ Lectures and final exam
  □ Course project

■ Basics
  □ Program analysis
  □ Deep learning