Analyzing Software using Deep Learning

Lecture 4:
Classifying Programs with Convolutional Networks

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

- Convolutional networks
  - Motivation and basics
  - Properties
  - Pooling

- Tree convolution for program classification

Based on "Convolutional Neural Networks over Tree Structures for Programming Language Processing" by Mou et al., 2016
Receptive fields of cats and monkeys

- Some neurons in visual cortex individually respond to small regions of the visual field
- Two visual cell types
  - Simple cells: Respond to straight edges with particular orientations
  - Complex cells: Sensitive to larger receptive field but insensitive to exact location of edges

Inspired work on neural network-based image recognition
Intuition

Input data is hierarchically organized

\( \rightarrow \) E.g., images or source code

Primitive features: Oriented edges

Object parts: wheels, windows

Object: car house

\( \rightarrow \) Image recognition
Convolutional Networks

- **Feedforward** neural network architecture
- Connectivity pattern **exploits hierarchical structure** of input data
- Not a fully connected network
- **Convolution function**: Mathematical approximation of stimuli within receptive field
- **Applications**:  
  - Image and video recognition  
  - Natural language processing  
  - Classification of programs
Complexity of **Fully Connected Networks**

**Suppose:**
- Input of length \( n \)
- Single output
- 3 hidden layers
- Fully connected

**Question:** How many weights does the network have for
\( n = 32 \times 32 = 1024 \)
(e.g., a small image)

\[
\begin{align*}
\text{n}^2 + \text{n}^2 + \text{n}^2 + \text{n} \\
= 3 \times 46752 \text{ weights} \\
= 1024^2 + 1024^2 + 1024^2 + 1024
\end{align*}
\]
- Each stored in memory
- Each optimized individually
Reducing Complexity via Convolution

Instead of fully connected computation graph:

- Each input influences at most \( k \) neurons
- Each neuron in the convoluted layer is activated by at most \( k \) neurons

\[ k = 3 \]
Kernel: Parameters for Convolution

\[ \begin{align*}
&\text{Convolution } S \\
&\text{Input } I
\end{align*} \]

\[ \begin{array}{c}
\text{Kernel } K \\
(\text{vector of length } k)
\end{array} \]

\( S \) is product of part of \( I \) and \( K \)
(We ignore the boundaries here.)

Example:

\[ \begin{align*}
K &= [2, 5, 3] \\
I &= [3, 1, 2, 3, 4]
\end{align*} \]

\[ S(2) = 1 \cdot 2 + 2 \cdot 5 + 3 \cdot 3 \]
Two-Dimensional Scenario

E.g., pixels of an image

\[ S(i,j) = \sum_{m} \sum_{n} I(i+m, j+n) \cdot K(m,n) \]

\[ S(0,0) = a \cdot w + b \cdot x + e \cdot y + f \cdot z \]
\[ S(0,1) = b \cdot w + c \cdot x + f \cdot y + g \cdot z \]
\[ S(1,2) = g \cdot w + h \cdot x + k \cdot y + l \cdot z \]
Properties

Three properties that help improve learning:

- Sparse interactions
- Parameter sharing
- Equivariant representations
Sparse Interactions

- Fewer connections than in fully connected network
  - Fewer weights to store
  - Fewer weights to optimize

- But: In deep networks (= many layers), neurons in deeper layers may indirectly interact with many inputs
Parameter Sharing

- Fully connected
  - Parameters to optimize: Weights in weight matrix
  - Each weight used for exactly one connection
  - Does not reduce time of forward propagation
  - But: reduces storage requirements

- Convolutional
  - Parameters to optimize: Weights in kernel matrix
  - Same kernel used for many parts of input
  - Connections all share the same kernel parameter
Equivariant Representations

Function $f(x)$ is equivariant to function $g$ if

$$f(g(x)) = g(f(x))$$

→ If input changes, then output changes in same way

Applications:

- Images: If object is moved, its representation will move by same amount

- Source code: If a statement with a particular property occurs anywhere, its representation remains the same
Big Picture

Convolution: Typically used in combination with other steps

Pooling

Detection

Convolution

Input

Non-linear activation function (e.g., rectified linear unit)

See before
Pooling

- Form of downsampling

- Replaces output at certain location with summary of nearby outputs

- Intuition: Exact location of a "feature" is less important than its presence and its location w.r.t. other features
Max Pooling

- Summarize region into the maximum output within this region

Example:

\[
\begin{array}{ccc}
1 & 0 & 2 & 3 \\
4 & 6 & 6 & 8 \\
3 & 1 & 0 & \\
1 & 2 & 2 & 4 \\
\end{array}
\]

Max pooling:

\[
\begin{array}{ccc}
6 & 8 & \\
3 & 4 & \\
\end{array}
\]
Plan for Today

- Convolutional networks
  - Motivation and basics
  - Properties
  - Pooling

- Tree convolution for program classification

Based on "Convolutional Neural Networks over Tree Structures for Programming Language Processing" by Mou et al., 2016
Convolution of Programs

- Convolution exploits hierarchical structure of input data
- Programs are hierarchical data
  - Coarse-grained level:
    Projects – Packages – Classes – Methods
  - Code level:
    Abstract syntax trees
- Idea: Use convolution to identify important features in code
Applications

Various possible applications

Today: Classification

- Various possible ways to classify programs
  - Project where code comes from
  - Author who wrote the code
  - Instances of bug patterns
  - Malware vs. benign code

- Here: Identify functionality of code
Overview

Program → Convolutional Neural Network → Softmax

Probabilities of different categories in classification
Tree Representation

- AST where each node has at most two children ("continuous binary tree")

Example:

```plaintext
int a = b + 3;
```

```
        Decl
        /   \
TypeDecl  Binary Op
     /     /   |
 Identifier Type   ID  Constant
```
Detailed Overview

AST → Representation Learning → Vector representation of AST nodes

Tree Convolution & Pooling → Fixed-size vector → Hidden layer → Softmax

Fully connected
Representation Learning

- Represent AST node as fixed-size vector ("embedding")

- Similar nodes should have similar vectors
  - E.g., While ≈ For but While & Constant

- Learned in separate pre-training step
Representation of Nodes

- Representation of P:
  \[ p_{rep} = W_{comb1} \cdot \text{vec}(p) + W_{comb2} \cdot \text{childrenVec}(p) \]

Goal: Parents encode information about children

\[ \text{vec}(p) \approx \text{childrenVec}(p) \]

\[ \text{tanh} \left( \text{left}\cdot W_{left} \cdot \text{vec}(c_i) + \text{right}\cdot W_{right} \cdot \text{vec}(c_2) + b_{rep} \right) \]

where \[ d_i = \frac{\# \text{leaves nodes under } c_i}{\# \text{leaves nodes under } p} \]

To minimize distance between vec(p) and childrenVec(p)
Example

```
Decl
  Type Decl
    | ID
  ID Type
    | ID Constant
```

Pre-learning

```
"Main" learning
```

```
W_comb1 \cdot vec("Binary Op")
+ W_comb2 \cdot \tanh\left( \frac{1}{2} \cdot W_{ep}^{mp} \cdot vec("ID") + \frac{1}{2} \cdot W_{right}^{mp} \cdot vec("Constant") + b_{mp} \right)
```