Outline

Introduction

Background
- Deep Neural Networks
- Real-Valued Representation Learning

Our Models
- Building Program Vector Representations for Deep Learning
- TBCNN: A Tree-based Convolutional Neural Network for Programming Language Processing

Conclusion and Discussion

Philosophy of Science: Also Belief

References
Introduction
Deep Neural Networks

- Widely applied machine learning architectures
  - speech recognition
  - computer vision
  - natural language processing
- Capable of capturing highly complicated (non-linear) features efficiently
- Very little human engineering and prior knowledge is required
  - people specify the model; machines learn details
[Hindle et al., 2012] compares programming languages to natural languages, and conclude that programs also have rich statistical properties:

- Difficult for human to capture
- Justifying learning-based approaches

However, no deep learning approaches have been proposed or applied in the field of program analysis.
Contributions of Our Work

- We are the first to apply deep learning to program analysis

- We propose a real-valued vector representation learning based on abstract syntax trees [Mou et al., 2014b]

- We propose a tree-based convolutional neural network to capture tree structural information [Mou et al., 2014a]
Background
Deep Neural Networks
A Single Layer of Neuron

Model:

\[ y = f(W \cdot x + b) \]

Training:

Gradient descent \( W \leftarrow W - \alpha \frac{\partial J}{\partial W}, \ b \leftarrow b - \alpha \frac{\partial J}{\partial b} \)

Limitation:

Linear separation
Model: Stacking multiple layers of neurons

Training: Gradient descent with back propagation
Multi-Layer Neural Networks

Model power:

- 2 layers for any Boolean or continuous function
- 3 layers for any function

Limitation:

- Inefficient (in terms of representation)
  The number of hidden units may grow exponentially to capture highly complicated features
- Poor generalization
  Too many parameters $\Rightarrow$ High VC dimension $\Rightarrow$ Poor generalization
Deep Neural Networks

- Efficient to capture highly complicated features
  Features are organized hierarchically, local features at lower layers and abstract features at higher layers

- Extremely difficult to train
  - Long term dependency (gradient would either vanish or blow up)
  - Local optima far from optimal
Deep Learning

Successful pretraining methods extract features unsupervisedly

- **Restricted Boltzmann Machine**
  
  Minimize the energy

- **Autoencoder**
  
  Minimize reconstruction error

2-stage strategy

1. Pretraining to initialize the weights meaningfully

2. Fine-tuning with back propagation so that the weights are specific to a problem
Real-Valued Representation Learning
Discrete Variables

Words are discrete!

They can't be fed to neural networks directly. (Recall $W \cdot x$)

Word 100 is 100x larger than Word 1?
The basic idea:

- Map each word to a vector in $\mathbb{R}^k$
- Each dimension capturing some (anonymous) feature
Learning Vector Representations

- [Bengio et al., 2003], maximizing the conditional probability of the $n$-th word given $n - 1$ words
- [Mnih and Hinton, 2007], maximizing the energy defined on neighboring words
- [Morin and Bengio, 2005, Mnih and Hinton, 2009], hierarchical architectures to reduce the computational cost
- [Collobert et al., 2011], negative sampling
- [Mikolov et al., 2010], recurrent neural network
The goal of language models: maximizing the joint probability of a corpus
Our Models
Building Program Vector Representations for Deep Learning
The Granularities of Program Analysis

- Characterize level?
- Token level?
- Nodes in Abstract Syntax Tree (AST)?
- Statement level? or higher?
double doubles(double doublee) {
    return 2 * doublee;
}

The Abstract Syntax Tree

```
double doubles(double doublee) {
    return 2 * doublee;
}
```

```
FuncDef
  Decl
    FuncDecl
      ParameterList
        Decl
          TypeDecl
            Decl
              TypeDecl
                IdentifierType
      TypeDecl
        IdentifierType
  Compound
    Return
      BinaryOp
        Constant
          ID
```

Lili Mou | SEKE Team
Formalization

The goal: To code parent’s representation by its children’s via a single layer of neurons

\[ \text{vec}(p) \approx \tanh \left( \sum_{i=1}^{n} l_i W_i \cdot \text{vec}(c_i) + b \right) \]

where \( l_i = \frac{\text{#leaves under } c_i}{\text{#leaves under } p} \) are the coefficients for \( W \)'s.
Define distance (Euclidean distance square)

\[ d = \left\| \text{vec}(p) - \tanh \left( \sum_{i=1}^{n} l_i W_i \cdot \text{vec}(c_i) + b \right) \right\|_2^2 \]

Cost function

\[ J(d^{(i)}, d_c^{(i)}) = \max \left\{ 0, \Delta + d^{(i)} - d_c^{(i)} \right\} \]

Training objective

\[ \min_{\Theta} \sum_i J(d^{(i)}, d_c^{(i)}) \]
**Empirical Results**

Examples of the nearest neighbor query results.

<table>
<thead>
<tr>
<th>Query</th>
<th>Most Similar</th>
<th>Results</th>
<th>Most Dissimilar</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>BinaryOp, Constant, ArrayRef, Assignment, StructRef</td>
<td>· · ·</td>
<td>PtrDecl, Compound, Root, Decl, TypeDecl</td>
</tr>
<tr>
<td>Constant</td>
<td>ID, UnaryOp, StructRef, ArrayRef, Cast</td>
<td>· · ·</td>
<td>EnumeratorList, ExprList, If, FuncDef, Compound</td>
</tr>
<tr>
<td>BinaryOp</td>
<td>ArrayRef, Assignment, StructRef, UnaryOp, ID</td>
<td>· · ·</td>
<td>PtrDecl, Compound, FuncDecl, Decl, TypeDecl</td>
</tr>
<tr>
<td>ArrayRef</td>
<td>BinaryOp, StructRef, UnaryOp, Assignment, Return</td>
<td>· · ·</td>
<td>Compound, PtrDecl, FuncDecl, Decl, TypeDecl</td>
</tr>
<tr>
<td>If</td>
<td>For, Compound, Break, While, Case</td>
<td>· · ·</td>
<td>BinaryOp, TypeDecl, Constant, Decl, ID</td>
</tr>
<tr>
<td>For</td>
<td>If, While, Case, Break, Struct</td>
<td>· · ·</td>
<td>BinaryOp, Constant, ID, TypeDecl, Decl</td>
</tr>
<tr>
<td>Break</td>
<td>While, Case, Continue, Switch, InitList</td>
<td>· · ·</td>
<td>BinaryOp, Constant, TypeDecl, Decl, ID</td>
</tr>
<tr>
<td>While</td>
<td>Switch, Continue, Label, Goto</td>
<td>· · ·</td>
<td>BinaryOp, Constant, Decl, TypeDecl, ID</td>
</tr>
<tr>
<td>FuncDecl</td>
<td>ArrayDecl, PtrDecl, FuncDef, Typename, Root</td>
<td>· · ·</td>
<td>ArrayRef, FuncCall, IdentifierType, BinaryOp, ID</td>
</tr>
<tr>
<td>ArrayDecl</td>
<td>FuncDecl, PtrDecl, Typename, FuncDef, While</td>
<td>· · ·</td>
<td>BinaryOp, Constant, FuncCall, IdentifierType, ID</td>
</tr>
<tr>
<td>PtrDecl</td>
<td>FuncDecl, Typename, FuncDef, ArrayDecl</td>
<td>· · ·</td>
<td>FuncCall, ArrayRef, Constant, BinaryOp, ID</td>
</tr>
</tbody>
</table>
### $k$-Means Clustering ($k = 3$)

<table>
<thead>
<tr>
<th>Cluster</th>
<th>Symbols</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>UnaryOp, FuncCall, Assignment, ExprList, StructRef, BinaryOp, ID, Constant, ArrayRef</td>
</tr>
<tr>
<td>2</td>
<td>FuncDef, Typedef, FuncDecl, Compound, ArrayDecl, PtrDecl, Decl, Root</td>
</tr>
<tr>
<td>3</td>
<td>Typedef, Struct, For, Union, CompoundLiteral, TernaryOp, Label, InitList, IdentifierType, Return, Enum, Break, DoWhile, Case, DeclList, Default, While, Continue, ParamList, Enumerator, Typename, Goto, Cast, Switch, EmptyStatement, EnumeratorList, If</td>
</tr>
</tbody>
</table>
Performance in Supervised Classification

(A) Learning curve of training

(B) Learning curve of CV
TBCNN: A Tree-based Convolutional Neural Network for Programming Language Processing
Motivation

Programs and natural languages are different in that

- Natural languages contain more symbols (words)
- Programs contain more structure information

“The dog the stick the fire burned beat bit the cat.” [Pinker, 1994]
Architecture of TBCNN

Vector representation and coding → Tree-based convolution → 3-way pooling → Hidden layer → Output layer → softmax
\[ p = W_{\text{comb1}} \cdot \text{vec}(p) \]
\[ + W_{\text{comb2}} \cdot \tanh \left( \sum_i l_i W_{\text{code},i} \cdot \text{vec}(x_i) + b_{\text{code}} \right) \]
Tree-based Convolution

\[ y = \tanh \left( \sum_{i=1}^{n} W_{\text{conv},i} \cdot x_i + b_{\text{conv}} \right) \]
The “Continuous Binary Tree” Model

\[ W_i = \eta_i^{(t)} W^{(t)} + \eta_i^{(l)} W^{(l)} + \eta_i^{(r)} W^{(r)} \]
Problem Definition

- POJ problems
- 2 groups, 4 problems in each groups
- Supervised multi-class classification according to program functionalities
<table>
<thead>
<tr>
<th>GRP.</th>
<th>Method</th>
<th>Train Err.</th>
<th>CV Err.</th>
<th>Test Err.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Random guess</td>
<td>75</td>
<td>75</td>
<td>75</td>
</tr>
<tr>
<td></td>
<td>LR</td>
<td>24.3</td>
<td>26.86</td>
<td>26.7</td>
</tr>
<tr>
<td></td>
<td>Linear SVM</td>
<td>24.89</td>
<td>27.51</td>
<td>28.48</td>
</tr>
<tr>
<td></td>
<td>RBF SVM</td>
<td>4.38</td>
<td>12.63</td>
<td>11.31</td>
</tr>
<tr>
<td></td>
<td>TBCNN</td>
<td>4.03</td>
<td>9.98</td>
<td>10.14</td>
</tr>
<tr>
<td></td>
<td>TBCNN+BOW</td>
<td>3.86</td>
<td>8.37</td>
<td><strong>8.53</strong></td>
</tr>
<tr>
<td>2</td>
<td>Random guess</td>
<td>75</td>
<td>75</td>
<td>75</td>
</tr>
<tr>
<td></td>
<td>LR</td>
<td>16.86</td>
<td>18.04</td>
<td>18.84</td>
</tr>
<tr>
<td></td>
<td>Linear SVM</td>
<td>17.18</td>
<td>17.87</td>
<td>19.48</td>
</tr>
<tr>
<td></td>
<td>RBF SVM</td>
<td>0.27</td>
<td>8.21</td>
<td>8.86</td>
</tr>
<tr>
<td></td>
<td>TBCNN</td>
<td>0.48</td>
<td>5.31</td>
<td>4.98</td>
</tr>
<tr>
<td></td>
<td>TBCNN+BOW</td>
<td>0.54</td>
<td>3.70</td>
<td><strong>3.70</strong></td>
</tr>
</tbody>
</table>
Detecting Bubble Sort

- Data
  
  109 source codes contain bubble sort
  109 source codes do not contain sort
  1:1 for developing and testing

- Training
  
  Generate \(~10000\) mock data samples

- Results

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Features</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rand/majority</td>
<td>–</td>
<td>50.0</td>
</tr>
<tr>
<td>RBF SVM</td>
<td>Bag-of-words</td>
<td>62.3</td>
</tr>
<tr>
<td>RBF SVM</td>
<td>Bag-of-trees</td>
<td>77.1</td>
</tr>
<tr>
<td>TBCNN</td>
<td>Learned</td>
<td>89.1</td>
</tr>
</tbody>
</table>
Conclusion and Discussion
Wrap up

- Deep learning and representations learning background
- Building program vector representations
- Tree-based convolutional neural networks
Philosophy of Science: Also Belief
Is computer science science?

Is political science science?

Discovery v.s. Invention
Research Pipeline

- Learning foundations
- Catching up the literature
- Figuring out new ideas
- Implementing your idea
- Experimenting for improvement
- Writing up
Thank you for listening!

Questions?
References
References


Lili Mou | SEKE Team


