Deep Learning for Program Analysis

Lili Mou

January, 2016

Outline

Introduction

- 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]

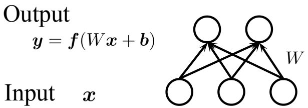
Background

Deep Neural Networks

A Single Layer of Neuron

Model:

$$\boldsymbol{y} = \boldsymbol{f}(W \cdot \boldsymbol{x} + \boldsymbol{b})$$



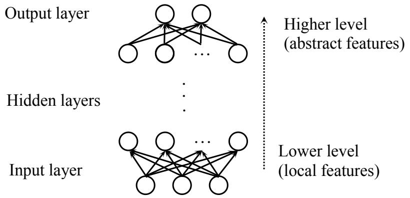
Training:

Gradient descent $W \leftarrow W - \alpha \frac{\partial J}{\partial W}$, $\boldsymbol{b} \leftarrow \boldsymbol{b} - \alpha \frac{\partial J}{\partial \boldsymbol{b}}$

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

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 $W \cdot \boldsymbol{x}$)

Word 100 is 100x larger than Word 1?

Real-Valued Representations

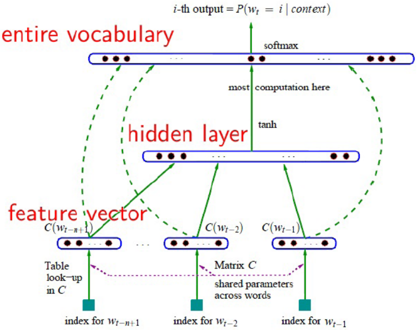
The basic idea:

- 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

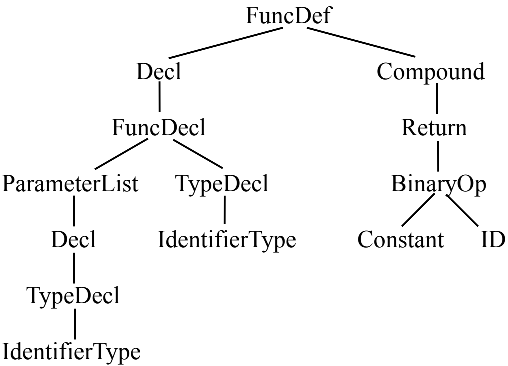
Building Program Vector Representations for Deep Learning

The Granularities of Program Analysis

- Characterize level?
- Token level?
- Nodes in Abstract Syntax Tree (AST)?
- Statement level? or higher?

The Abstract Syntax Tree

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



Formalization

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

$$\mathrm{vec}(p) \approx \tanh\left(\sum_{i=1}^{n} l_i W_i \cdot \mathrm{vec}(c_i) + \boldsymbol{b}\right)$$

where $l_i = \dfrac{\#\text{leaves under } c_i}{\#\text{leaves under } p}$ are the coefficients for W's.

Negative Sampling

Define distance (Euclidean distance square)

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

Cost function

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

Training objective

$$\underset{\Theta}{\text{minimize}} \sum_i J(d^{(i)}, d_c^{(i)})$$

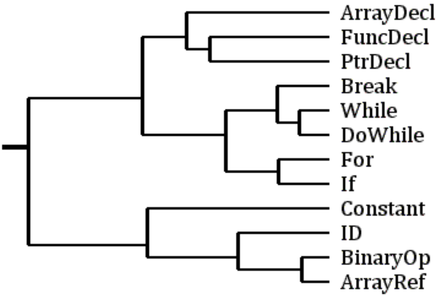
Examples of the nearest neighbor query results.

Query	Results		
	Most Similar		**Most Dissimilar**
ID	BinaryOp, Constant, ArrayRef, Assignment, StructRef	···	PtrDecl, Compound, Root, Decl, TypeDecl
Constant	ID, UnaryOp, StructRef, ArrayRef, Cast	···	EnumeratorList, ExprList, If, FuncDef, Compound
BinaryOp	ArrayRef, Assignment, StructRef, UnaryOp, ID	···	PtrDecl, Compound, FuncDecl, Decl, TypeDecl
ArrayRef	BinaryOp, StructRef, UnaryOp, Assignment, Return	···	Compound, PtrDecl, FuncDecl, Decl, TypeDecl
If	For, Compound, Break, While, Case	···	BinaryOp, TypeDecl, Constant, Decl, ID
For	If, While, Case, Break, Struct	···	BinaryOp, Constant, ID, TypeDecl, Decl
Break	While, Case, Continue, Switch, InitList	···	BinaryOp, Constant, TypeDecl, Decl, ID
While	Switch , Continue , Label , Goto	···	BinaryOp, Constant, Decl, TypeDecl, ID
FuncDecl	ArrayDecl, PtrDecl, FuncDef, Typename, Root	···	ArrayRef, FuncCall, IdentifierType, BinaryOp, ID
ArrayDecl	FuncDecl, PtrDecl, Typename, FuncDef, While	···	BinaryOp, Constant, FuncCall, IdentifierType, ID
PtrDecl	FuncDecl, Typename, FuncDef, ArrayDecl	···	FuncCall, ArrayRef, Constant, BinaryOp, ID

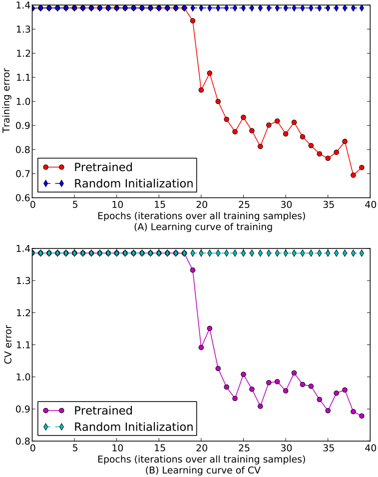
k-Means Clustering ($k = 3$)

Cluster	Sybmols
1	UnaryOp, FuncCall, Assignment, ExprList, StructRef, BinaryOp, ID, Constant, ArrayRef
2	FuncDef, TypeDecl, FuncDecl, Compound, ArrayDecl, PtrDecl, Decl, Root
3	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

Hierarchical Clustering



Performance in Supervised Classification



TBCNN: A Tree-based Convolutional Neural Network for Programming Language Processing

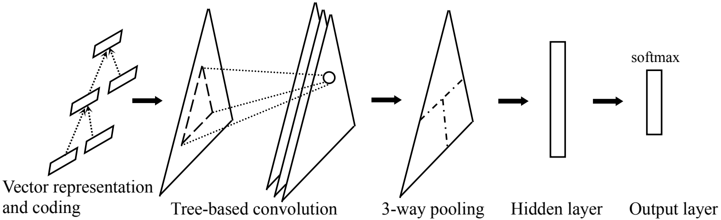
Motivation

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]

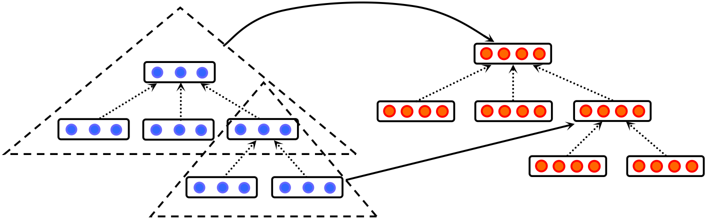
Architecture of TBCNN

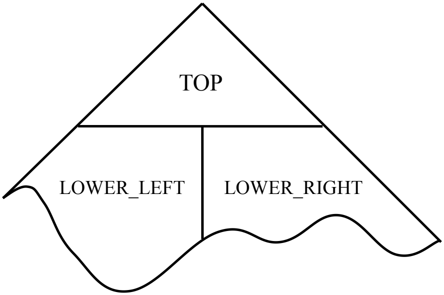


$$
\begin{aligned}
\boldsymbol{p} = W_{\mathsf{comb1}} \cdot \mathrm{vec}(p) \\
+ W_{\mathsf{comb2}} \cdot \tanh\left(\sum\nolimits_i l_i W_{\mathsf{code},i} \cdot \mathrm{vec}(x_i) + \boldsymbol{b}_{\mathsf{code}}\right)
\end{aligned}
$$

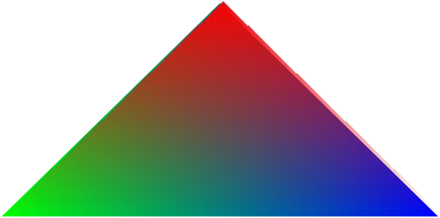
Tree-based Convolution

$$\boldsymbol{y} = \tanh\left(\sum_{i=1}^{n} W_{\mathsf{conv},i} \cdot \boldsymbol{x}_i + \boldsymbol{b}_{\mathsf{conv}}\right)$$




TOP
LOWER_LEFT
LOWER_RIGHT

$$W_i = \eta_i^{(t)} W^{(t)} + \eta_i^{(l)} W^{(l)} + \eta_i^{(r)} W^{(r)}$$



Problem Definition

- POJ problems
- 2 groups, 4 problems in each groups
- Supervised multi-class classification according to program functionalities

GRP.	Method	Train Err.	CV Err.	Test Err.
1	Random guess	75	75	75
	LR	24.3	26.86	26.7
	Linear SVM	24.89	27.51	28.48
	RBF SVM	4.38	12.63	11.31
	TBCNN	4.03	9.98	10.14
	TBCNN+BOW	3.86	8.37	**8.53**
2	Random guess	75	75	75
	LR	16.86	18.04	18.84
	Linear SVM	17.18	17.87	19.48
	RBF SVM	0.27	8.21	8.86
	TBCNN	0.48	5.31	4.98
	TBCNN+BOW	0.54	3.70	**3.70**

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 $\sim$10000 mock data samples

- Results

Classifier	**Features**	**Accuracy**
Rand/majority	–	50.0
RBF SVM	Bag-of-words	62.3
RBF SVM	Bag-of-trees	77.1
TBCNN	Learned	**89.1**

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

Questions?

References

References

[Bengio et al., 2003] Bengio, Y., Ducharme, R., Vincent, P., and Jauvin, C. (2003). A neural probabilistic language model. Journal of Machine Learning Research, 3:1137–1155.

[Collobert et al., 2011] Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P. (2011). Natural language processing (almost) from scratch. The Journal of Machine Learning Research, 12:2493–2537.

[Hindle et al., 2012] Hindle, A., Barr, E., Su, Z., Gabel, M., and Devanbu, P. (2012). On the naturalness of software. In Proceedings of 34th International Conference on Software Engineering.

[Mikolov et al., 2010] Mikolov, T., Karafiat, M., Burget, L., Cernocky, J., and Khudanpur, S. (2010). Recurrent neural network based language model. In INTERSPEECH.

[Mnih and Hinton, 2007] Mnih, A. and Hinton, G. (2007). Three new graphical models for statistical language modelling. In Proceedings of the 24th International Conference on Machine learning.

[Mnih and Hinton, 2009] Mnih, A. and Hinton, G. (2009). A scalable hierarchical distributed language model. In Advances in Neural Information Processing Systems.

[Morin and Bengio, 2005] Morin, F. and Bengio, Y. (2005). Hierarchical probabilistic neural network language model. In Proceedings of International Conference on Artificial Intelligence and Statistics.

[Mou et al., 2014a] Mou, L., Li, G., Jin, Z., Zhang, L., and Wang, T. (2014a). Tbcnn: A tree-based convolutional neural network for programming language processing. arXiv preprint arXiv:1409.5718.

[Mou et al., 2014b] Mou, L., Li, G., Liu, Y., Peng, H., Jin, Z., Xu, Y., and Zhang, L. (2014b). Building program vector representations for deep learning. arXiv preprint arXiv:1409.3358.

[Pinker, 1994] Pinker, S. (1994). The Language Instinct: The New Science of Language and Mind. Pengiun Press.