Neural Machine Translation Inspired Binary Code Similarity Comparison *beyond Function Pairs*

CoRR 2018
Fei Zuo, Xiaopeng Li, Zhexin Zhang, Patrick Young, Lannan Luo, Qiang Zeng

김영철

2019. 4. 4.

Introduction

- P1 : Given a pair of BBs of different ISAs, determining whether their semantics is similar
  - propose to learn from techniques of Neural Machine Translation (NMT)

- P2 : Given a piece of critical code, determining whether it is contained in another piece of code of a different ISA
  - propose techniques for cross-architecture basic block comparison

Related Work

- Traditional Code Similarity Comparison
  - Mono-architecture solutions
    - Static plagiarism detection or clone detection
    - Dynamic birthmark based approaches
  - Cross-architecture solutions
    - fuzzing-based basic block similarity comparison and graph matching-based algorithm [52]
      - are too expensive to handle a large number of function pairs.
    - Esh & its successor [15]
      - define strand (data-flow slices of basic blocks) as the basic comparable unit
      - useSMT solver to verify function similarity

Related Work

- Machine Learning-based Code Similarity Comparison
  - Mono-architecture solutions
    - to detect the similarity between source code fragments [63]
    - introduce a tree-based convolutional neural network based on program abstract syntax tress to detect similar source code snippets [47]
    - to identify buggy source code files [25] [26]

  - Cross-architecture solutions
    - [19] [65] are prior state-of-the-art works on cross-architecture bug search
      - use machine learning & deep learning to convert CFGs of functions into vectors for similarity comparison
    - [8] introduces a selective inlining technique to capture the function semantics and extracts partial traces of various lengths to model functions

Overview

- Given a query binary code component Q,
  its objective is to find programs that contain Q or the similar.
  - three layers of code semantics
    (1) neural network-based cross-lingual basic-block embedding model
    (2) the path similarity using the LCS algorithm to compare two paths
    (3) the component similarity explores multiple path pairs

Overview

- Basic Block Similarity Detection
    - The key is to measure the similarity of two blocks, regardless of ISAs.
    - Siamese architecture [6] with each side employing the LSTM [24]

    - The model converts each block into an embedding

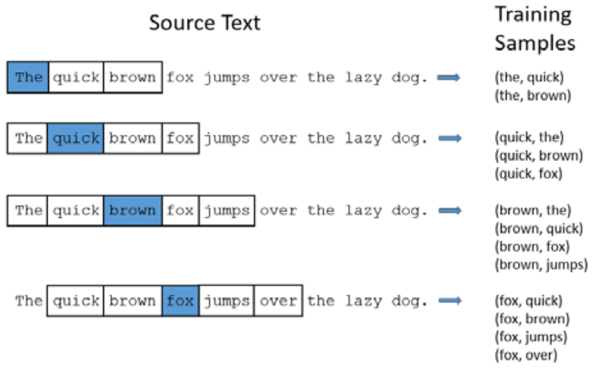
        (1) Instruction embedding generation
        (2) Basic-block embedding generation
        (3) Two block's similarity is the distance between their block embeddings

    - the model inherited from NMT automatically learn useful features
        - [19] [65], which manually select features, largely loses the information

Instruction Embedding Generation

- Background: Word Embedding
  - represent words in a high-dimensional space
  - is to capture the contextual semantic meaning of the word

  - [42]'s *skip-gram model* (현재 주어진 단어 주위 단어들의 등장 여부 유추)
    - a sliding windows is employed on a text stream
    - starts with a random vector for each word
    - with *negative sampling model* (일부만 뽑아서 계산)
    - can be trained at billions of words per hour
    - is entirely *unsupervised*



Instruction Embedding Generation

- Challenges
  (1) It have to train an instruction embedding model from scratch
  (2) the embedding generation for out-of-vocabulary (OOV) will fail

Instruction Embedding Generation

- Building Training Dataset
  - Preprocessing Training Data

    Rules to resolve the OOV problem

    (1) The number constant values are replaced with 0, and the signs are preserved

    (2) The string literals are replaced with <STR>

    (3) The function names are replaced with FOO

    (4) Other symbol constants are replaced with <TAG>

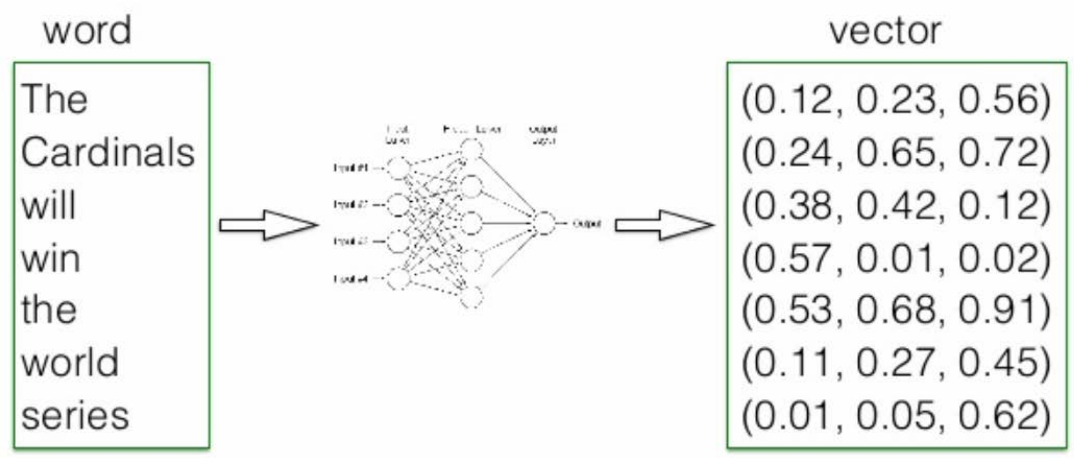
    ➔ can reduce many OOV cases

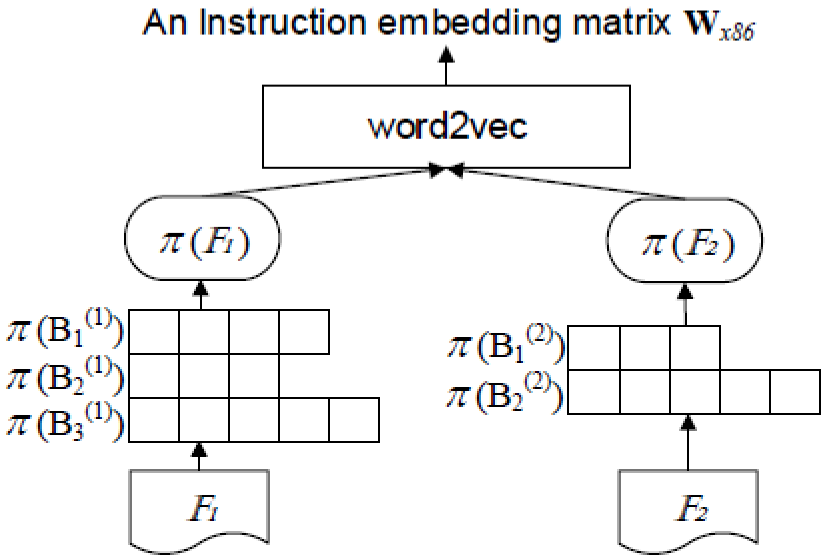
```
MOVL %ESI, $.L.STR.31
MOVL %EDX, $3
MOVQ %RDI, %RAX
CALLQ STRNCMP
TESTL %EAX, %EAX
JE .LBB0_5
```

```
MOVL ESI, <STR>
MOVL EDX, 0
MOVQ RDI, RAX
CALLQ FOO
TESTL EAX, EAX
JE <TAG>
```

Instruction Embedding Generation

- Training Instruction Embedding Model
  - ~~Word2Vec 를 이용해 학습 모델을 구현한다는데, Word2Vec 가 구체적으로 무엇인지 모르겠음.~~





Block Embedding Generation

- The most simple generating the embedding of a basic block is to compose all embeddings of the instruction
  - but cannot handle the cross-architecture differences

Source code

```
numblocks = (tmp_len+blocksize-1)/blocksize;
if(numblocks > pre_comp->numblocks)
```

X86-64 assembly

```
movq %rsi,80(%rsp)
addq %rax,%rsi
addq %rax,$-1
xorl %edx,%edx
divq %rsi
movq %rdx,96(%rsp)
cmpq %rax,16(%rdx)
jbe  .LBB2_68
```

ARM assembly

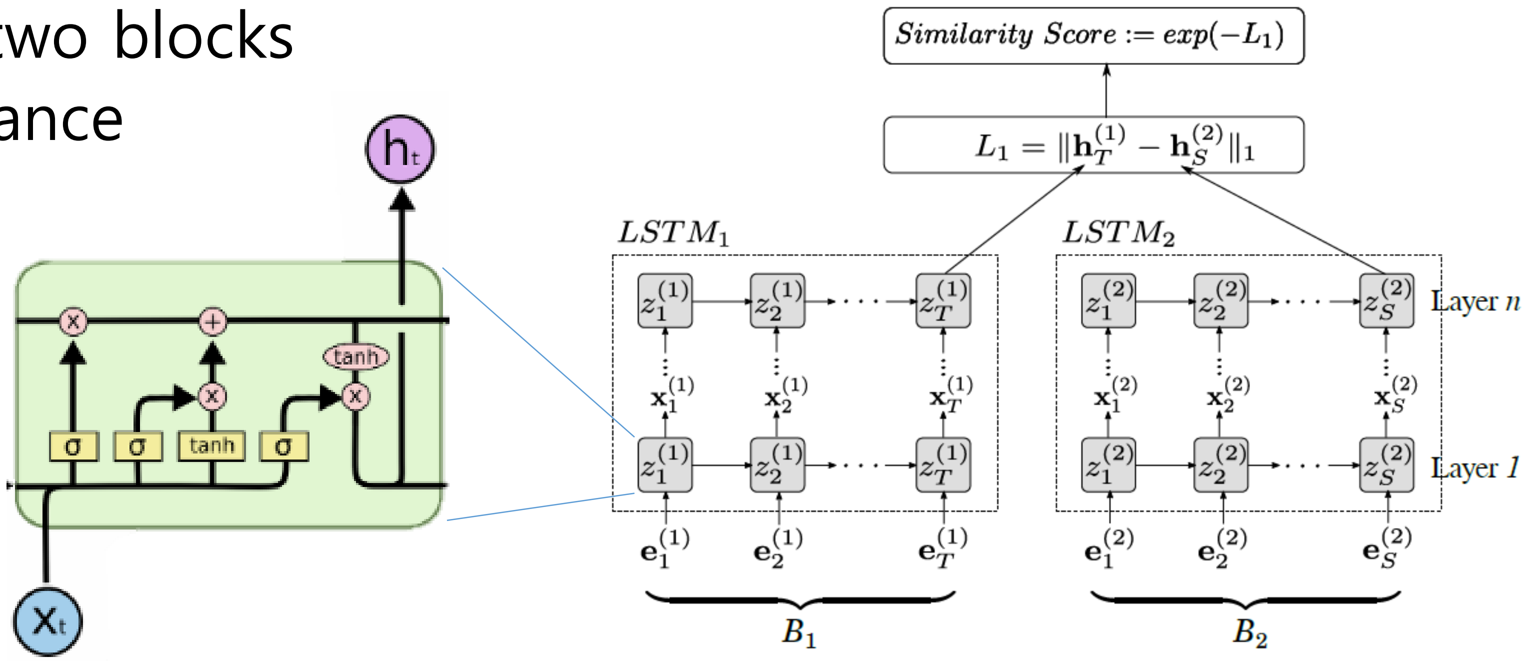
```
adds r1, r2, r1
adc  r7, r3, r0
subs r0, r1, #1
sbc  r1, r7, #0
bl   __udivdi3
ldr  r3, [sp, #60]
ldr  r2, [r3, #16]
ldr  r3, [r3, #20]
subs r2, r2, r0
sbcs r2, r3, r1
bhs  .LBB2_120
```

Block Embedding Generation

- Background: LSTM in NLP
  - LSTM is developed to address the difficulty of capturing long term memory in the basic RNN.
  - At each time step, a new word is encoded and the word dependencies embedded in the vector are updated
  - the final semantic vector can be viewed as a feature representation of the whole sentence.

Block Embedding Generation

- Cross-lingual Basic-block Embedding Model Architecture
  - design the model as a Siamese architecture [6] with each side employing the identical LSTM
  - The inputs are two blocks
  - Manhattan Distance



Block Embedding Generation

- Challenges
  - Two main challenges for block embeddings

    (1) In order to train, a large dataset containing labeled similar and dissimilar block pairs is needed

    ➔ 함수 단위로 비교할 때에는 함수 이름을 이용하여 ground truth 설정 가능

    ➔ 하지만, 블록 단위로 비교할 때에는 답을 알 수가 없음

    (2) The parameter values selected for NMT are not necessarily applicable to our mode, and need to be comprehensively examined.

Block Embedding Generation

- Building Dataset

  1) Generating Similar Basic-Block Pairs
     - modify various architecture-dependent LLVM backends

       베이직 블록 경계를 표시하고, 같은 소스 코드로 만들어진 어셈블리 블록에 동일한 ID로 마킹함

       여러 오프 소스 프로젝트를 이용하여 많은 수의 베이직 블록을 생성함.

  2) Generating Dissimilar Basic-Block Pairs
     - 같은 ID를 갖는 블록은 무조건 같은 블록인데 ID가 다르다고 무조건 다르지 않음.

       동일한 아키텍처 간 두 블록의 유사도를 n-gram 을 이용하여 측정

       ex. $B_1^{ARM} == B_1^{x86}$, $B_1^{x86} != B_2^{x86}$ ➔ $B_1^{ARM} != B_2^{x86}$

Path/Code Component Similarity Comparison

- 기존 연구는 single-architecture 조건이나 함수 쌍을 비교하는 연구를 수행함.
  - 치명적인 코드 조각이 삽입되면 탐지가 불가능 함.
  - 코드 조각 Q 의 CFG 를 path 로 분해하고 타겟 프로그램의 path 들과 비교
    - LCS 알고리즘을 이용한 유사도 계산
    - 타겟 프로그램에서 Q와 비슷한 것을 탐지

Path/Code Component Similarity Comparison

- Path Similarity Comparison
  - A linearly independent path
    - is a path that introduces at least one new node that is not included in any previous linearly independent paths [62]
  - CoP [37]와 비슷한 방법으로 path similarity 측정.
    - CoP 에서는 블록 유사도를 Symbolic Execution을 이용하지만 여기서의 방법은 훨씬 빠름
    - 타겟 프로그램 T에 있는 모든 linearly independent path 와 코드 조각 P의 path를 비교해서 가장 높은 유사도 값을 찾음

$$\psi(\mathcal{P}, T) = \frac{\max_{\mathcal{P}_i^t \in \Gamma} |\mathrm{LCS}(\mathcal{P}, \mathcal{P}_i^t)|}{|\mathcal{P}|}$$

Path/Code Component Similarity Comparison

- Component Similarity Comparison
  - Challenge
    - It is possible for code component to be inserted into the middle of a function
    - It is important to determine the correct starting points
    - This is a unique challenge compared to function-level code similarity comparison

Path/Code Component Similarity Comparison

- Component Similarity Comparison
  - Idea
    - store the embeddings of all basic blocks of the target program T
    - find a semantically equivalent block with the first basic block of Q
    - if find one or more blocks, do path exploration on each of them

Pair 1		Pair 2		Pair 3	
x86	*ARM*	*x86*	*ARM*	*x86*	*ARM*
MOVSLQ RSI,EBP	LDRB R0,[R8+R4]	MOVQ RDX,<TAG>+[RIP+0]	LDR R2,[R8+0]	MOVQ [RSP+0],RBX	LDR R0,[SP+0]
MOVZBL ECX,[R14,RBX]	STR R9,[SP]	MOVQ RDI,R12	MOV R0,R4	MOVQ [RSP+0],R14	STR R9,[SP+0]
MOVL EDX,<STR>	STR R0,[SP+0]	MOVL ESI,R14D	MOV R1,R5	ADDQ RDI,0	STR R0,[SP+0]
XORL EAX,EAX	ASR R3,R7,0	CALLQ FOO	BL FOO	CALLQ FOO	ADD R0,R1,0
MOVQ RDI,R13	MOV R0,R6	MOVQ RDI,R12	MOV R0,R4	MOVL ESI,<TAG>	BL FOO
CALLQ FOO	MOV R2,R7	CALLQ FOO	BL FOO	MOVQ RDI,[R12]	LDR R7,<TAG>
TESTL EAX,EAX	BL FOO	MOVQ RDX,<TAG>+[RIP+0]	LDR R2,[R8+0]	MOVB [RDI+0],AL	LDR R1,[R6]
JLE <TAG>	CMP R0,0	MOVQ RDI,R12	MOV R0,R4	CMPB [RDI+0],0	LDR LR,[SP+0]
	BLT <TAG>	MOVL ESI,R14D	MOV R1,R5	JNE <TAG>	MOV R12,R7
		CALLQ FOO	BL FOO		STRB R0,[R1+0]
		TESTL EAX,EAX	CMP R0,0		B <TAG>
		JNE <TAG>	BNE<TAG>		

<Examples of similar BB pairs that are correctly classified>

Pair 4		Pair 5		Pair 6	
x86	*ARM*	*x86*	*ARM*	*x86*	*ARM*
IMULQ R13,RAX,0	MOV R1,R0	XORL R14D,R14D	LDMIB R5,R0,R1	MOVL [RSP+0],R14D	SUB R2,R1,0
XORL EDX,EDX	LDR R6,[SP+0]	TESTQ RBP,RBP	CMP R0,R1	MOVQ RAX,[RSP+0]	MOV R10,0
MOVQ RBP,[RSP+0]	CMP R0, 0	JE <TAG>	BHS <TAG>	CMPB [RAX],0	CMP R2,0
DIVQ RBP	BEQ <TAG>			MOVQ [RSP+0],R13	MOV R9,0
JMP <TAG>				MOVQ [RSP+0],R15	BHI <TAG>
				JNE <TAG>	

<Examples of dissimilar BB pairs that are correctly classified>

E N D