Tree-Structured Capsule Networks for Program Source Code Processing

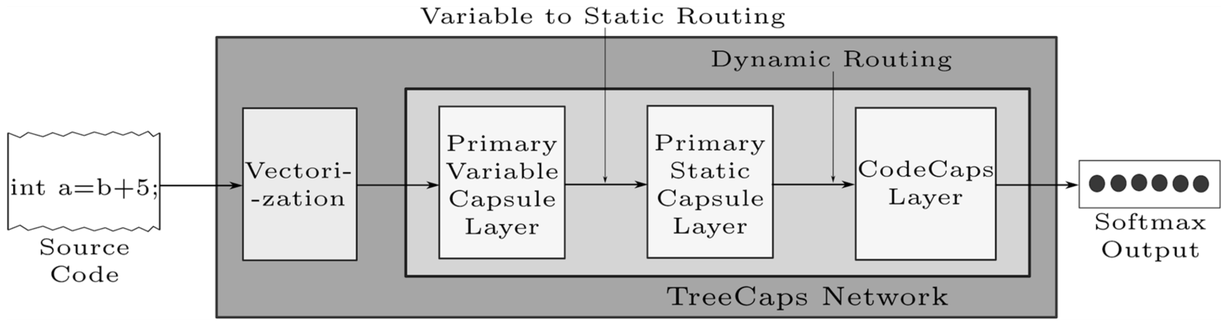
Vinoj Jayasundara[1,2], Nghi Duy Quoc Bui[2], Lingxiao Jiang[2], David Lo[2]

[1]A*STAR, Singapore [2]School of Information Systems, Singapore Management University

MOTIVATION

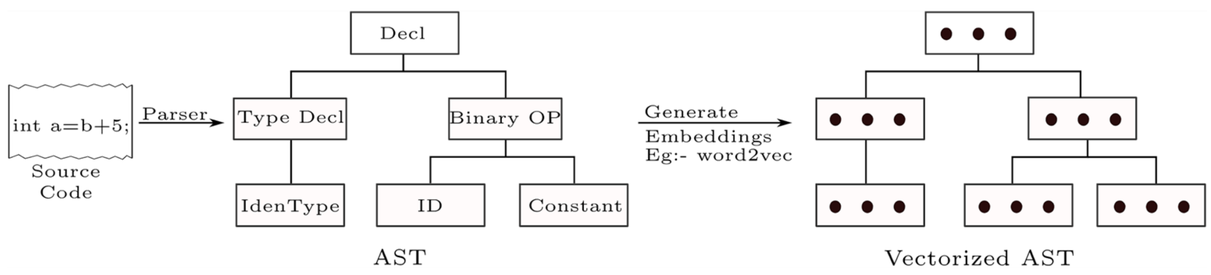
- Understanding program code is a fundamental step for many software engineering tasks.
- The existing approaches do not explicitly learn the dependency relationships present in source codes, hindering performance.
- We propose TreeCaps, which can automatically learn dependency relationships with the proposed variable to static routing algorithm.

TREECAPS OVERVIEW



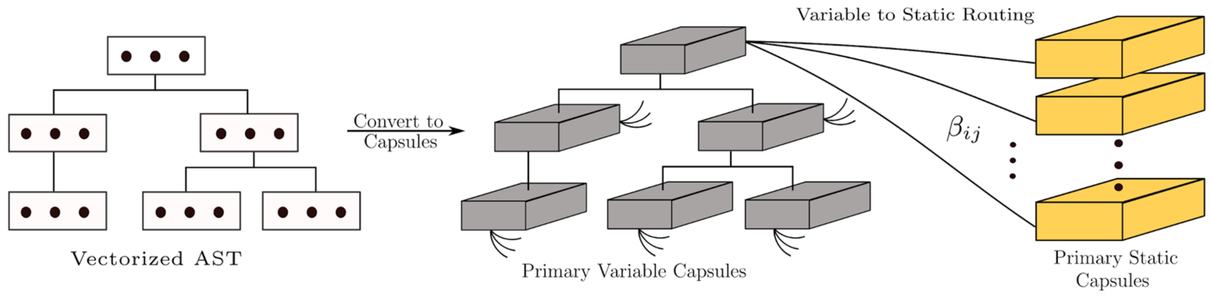
- The source code of the training sample program is parsed into an AST and vectorized with the aid of a suitable technique. (Eg :- Word2Vec)
- The vectorized AST is then fed to the proposed TreeCaps network, which consists of :
  - Primary Variable Capsule layer
    - ↕ Variable to Static Routing algorithm
  - Primary Static Capsule layer
    - ↕ Dynamic Routing algorithm
  - Code Capsule layer

ABSTRACT SYNTAX TREE VECTORIZATION



- Every raw source code is parsed with an appropriate parser corresponding to the programming language to generate the Abstract Syntax Tree (AST).
- We use ASTs to train the embeddings by using techniques similar to Penget al. (2015), which learns a vectorized vocabulary of node types.
- The learned vocabulary can subsequently be used to vectorize each individual node of the ASTs, generating the vectorized ASTs.

PRIMARY VARIABLE CAPSULE LAYER



Tree – Based Convolution: $\boldsymbol{y} = tanh(\sum_{i=1}^{K+1}[\eta_i^t \boldsymbol{W}^t + \eta_i^l \boldsymbol{W}^l + \eta_i^r \boldsymbol{W}^r]\boldsymbol{x}_i + \boldsymbol{b})$

- **y** corresponds to the output of one convolutional slice. We use ε such slices with different initializations for **W, b**. $\eta_i^t, \eta_i^l, \eta_i^r$ are weights defined corresponding to the depth and the position of the children nodes.
- We group the convolutional slices together to form sets of capsules with outputs $u_i \in \mathbb{R}^D$, where *D* is the dimensions of the capsules in the PVC layer.
- To vectorize each capsule output, we apply a non-linear *squash* function, producing the output of the PVC layer.

VARIABLE TO STATIC ROUTING ALGORITHM

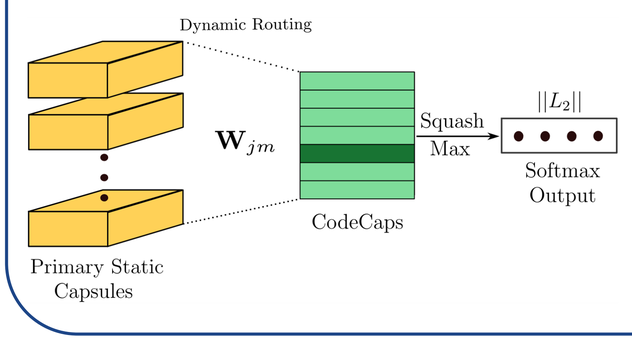
- The key issue with passing the outputs of the PVC layer to the Code Capsule layer is that the number of capsules vary with the training sample.
- To route capsules, we need to project them to a higher dimension with a transformation matrix learning dependency relations, which cannot be defined with variable dimensions. Thus, dynamic routing cannot be applied.

Solution : Proposed Variable to Static Routing Algorithm

Algorithm 1 Variable-to-Static Capsule Routing

procedure ROUTING($\hat{\mathbf{u}}_i, r, a, b$)
  $\hat{\mathbf{U}}_{\mathbf{sorted}} \leftarrow sort([\hat{\mathbf{u}}_1, ..., \hat{\mathbf{u}}_{N_{pvc}}])$
  Initialize $\mathbf{v}_j : \forall i, j \leq a, \mathbf{v_j} \leftarrow \hat{\mathbf{U}}_{\mathbf{sorted}}[i]$
  Initialize $\alpha_{ij} : \forall j \in [1, a], \forall i \in [1, b], \alpha_{ij} \leftarrow 0$
  for r iterations **do**
    $\forall j \in [1, a], \forall i \in [1, b], f_{ij} \leftarrow \hat{\mathbf{u}}_i \cdot \mathbf{v}_j$
    $\forall j \in [1, a], \forall i \in [1, b], \alpha_{ij} \leftarrow \alpha_{ij} + f_{ij}$
    $\forall i \in [1, b], \boldsymbol{\beta}_i \leftarrow Softmax(\boldsymbol{\alpha}_i)$
    $\forall j \in [1, a], \mathbf{s}_j \leftarrow \sum_i \beta_{ij} \hat{\mathbf{u}}_i$
    $\forall j \in [1, a], \mathbf{v}_j \leftarrow Squash(\mathbf{s}_j)$
  return $\mathbf{v}_j$

CODE CAPSULE LAYER



- This acts as the classification capsule layer.
- Since the output of the PSC layer is a fixed set of capsules, it can be routed to the CC layer with dynamic routing.

EXPERIMENTS AND RESULTS

- The means and the standard deviations from 3 trials are shown.

Model	Dataset A (Python)	Dataset B (Java)	Dataset C (C)
GGNN	-	85.00%	86.52%
TBCNN	99.30%	75.00%	79.40%
TreeCaps	100.00 ± 0.00%	92.11±0.90%	87.95±0.23%
TreeCaps (3-ens.)	**100.00%**	**94.08%**	**89.41%**

MODEL ANALYSIS

Model Variant	Accuracy
Variable-to-Static Routing Algorithm → Dynamic Pooling	83.43%
Instantiation parameters → D_{cc}= 4	90.90%
D_{cc}= 8	92.10%
D_{cc}= 12	90.33%
D_{cc}= 16	91.51%
TreeCaps → TreeCaps + Secondary Capsule Layer	92.31%
TreeCaps with Variable-to-Static Routing and D_{cc}= 8	**92.11%**

- The instantiation parameters D_{cc} of the CC layer acts as the dimensionality of the latent representation of source code.
  - D_{cc} ↑↑ - Sparsity and/or correlated instantiation parameters
  - D_{cc} ↓↓ - Under-representation

CONCLUSION

- TreeCaps learns rich syntactical structures and semantic dependencies in program source code.
- TreeCaps significantly outperforms the existing approaches on program classification robustly across programming languages.