TreeCaps: Tree-Structured Capsule Networks for Program Source Code Processing

Presented by Vinoj Jayasundara

Software Intelligence Group

December 1, 2019
1. Introduction to Capsule Networks

2. Methodology

3. Task: Program Classification

4. Limitations and Future Work
Capsule Network : Instantiation parameters

Capsule Networks can encode any entity in instantiation parameters.

Convolutional Networks

- Probability: 1.3

Part whole relationship

- A 6-dimensional capsule corresponding to a rectangle

Capsule Networks

- Color
- Angle (x-axis)
- Length
- Breadth
- Deformation
- Texture
Capsule Networks propose a novel routing by agreement algorithm.

![Diagram of Capsule Network](image)
1. Introduction to Capsule Networks

2. Methodology

3. Task: Program Classification

4. Limitations and Future Work
• 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 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 AST\(^1\).

\(^1\)Python - Python AST parser, C & Java - srcML
Abstract Syntax Tree Vectorization

- Every raw source code is parsed with an appropriate parser corresponding to the programming language to generate the AST\(^1\).

- We use ASTs to train the embeddings by using techniques similar to Penget al. (2015), which learns a vectorized vocabulary of node types.

---

\(^1\)Python - Python AST parser, C & Java - srcML
Every raw source code is parsed with an appropriate parser corresponding to the programming language to generate the AST\(^1\).

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.

\(^1\)Python - Python AST parser, C & Java - srcML
One of the main challenges in creating a tree-based capsule network is that the input of the network is tree-structured.

- Image data \( \in \mathbb{R}^{H \times W \times C} \), where \( H, W, C \) are fixed.
- Natural language data \( \in \mathbb{R}^{L \times E} \), where \( L, E \) are the fixed.
- Tree-structured data \( \in \mathbb{R}^{T \times V} \), where \( T \) is dynamic.
One of the main challenges in creating a tree-based capsule network is that the input of the network is tree-structured.

- ✔ Image data $\in \mathbb{R}^{H \times W \times C}$, where $H$, $W$, $C$ are fixed.
- ✔ Natural language data $\in \mathbb{R}^{L \times E}$, where $L$, $E$ are fixed.
- ✗ Tree-structured data $\in \mathbb{R}^{T \times V}$, where $T$ is dynamic.

A further challenge is that the number of children varies from node to node.
TreeCaps : Tree Structured Convolution

• One of the main challenges in creating a tree-based capsule network is that the input of the network is tree-structured.
  ✓ Image data $\in \mathbb{R}^{H \times W \times C}$, where $H, W, C$ are fixed.
  ✓ Natural language data $\in \mathbb{R}^{L \times E}$, where $L, E$ are fixed.
  × Tree-structured data $\in \mathbb{R}^{T \times V}$, where $T$ is dynamic.

• A further challenge is that the # of children varies from node to node.

Solutions :
• Zero padding ?
TreeCaps: Tree Structured Convolution

- One of the main challenges in creating a tree-based capsule network is that the input of the network is tree-structured.
  - ✓ Image data $\in \mathbb{R}^{H \times W \times C}$, where $H, W, C$ are fixed.
  - ✓ Natural language data $\in \mathbb{R}^{L \times E}$, where $L, E$ are the fixed.
  - × Tree-structured data $\in \mathbb{R}^{T \times V}$, where $T$ is dynamic.

- A further challenge is that the # of children varies from node to node.

Solutions:
- Zero padding ?
- Tree-based Convolution better

$$y = \tanh\left(\sum_{i=1}^{K+1} [\eta^t_i W^t + \eta^l_i W^l + \eta^r_i W^r] x_i + b\right) (1)$$

$\eta^t_i, \eta^l_i, \eta^r_i$ are weights defined corresponding to the depth and the position of the children nodes, and $Y_{conv} \in \mathbb{R}^{T \times V'}$. 

Vinoj Jayasundara  
TreeCaps: Tree-Structured Capsule Networks  
December 1, 2019 8 / 22
• \( y \) obtained from Eq 1 corresponds to the output of one convolutional slice. We use \( \varepsilon \) such slices with different initializations for \( \mathbf{W}, \mathbf{b} \).
• \( y \) obtained from Eq 1 corresponds to the output of one convolutional slice. We use \( \varepsilon \) such slices with different initializations for \( \mathbf{W}, \mathbf{b} \).

• We group the convolutional slices together to form \( N_{pvc} = \frac{T \times V' \times \varepsilon}{D_{pvc}} \) sets of capsules with outputs \( \mathbf{u}_i \in \mathbb{R}^{D_{pvc}}, i \in [1, N_{pvc}] \), where \( D_{pvc} \) is the dimensions of the capsules in the PVC layer.
• $y$ obtained from Eq 1 corresponds to the output of one convolutional slice. We use $\varepsilon$ such slices with different initializations for $W, b$.

• We group the convolutional slices together to form $N_{pvc} = \frac{T \times V' \times \varepsilon}{D_{pvc}}$ sets of capsules with outputs $u_i \in \mathbb{R}^{D_{pvc}}$, $i \in [1, N_{pvc}]$, where $D_{pvc}$ is the dimensions of the capsules in the PVC layer.

• To vectorize each capsule output, we subsequently apply a non-linear squash function, producing the output of the PVC layer $\in \mathbb{R}^{N_{pvc} \times D_{pvc}}$. 
• The key issue with passing the outputs of the PVC layer to the Code Capsule layer is that $N_{pvc}$ is variable with the training sample.
The key issue with passing the outputs of the PVC layer to the Code Capsule layer is that $N_{pvc}$ is variable with the training sample.

Prior to routing the lower level capsules to a set of higher level capsules, the lower dimensional capsule outputs need to be projected to the higher dimensionality, with a transformation matrix which learns the part-whole relationship between the lower and the higher level capsules.
The key issue with passing the outputs of the PVC layer to the Code Capsule layer is that $N_{pvc}$ is variable with the training sample.

Prior to routing the lower level capsules to a set of higher level capsules, the lower dimensional capsule outputs need to be projected to the higher dimensionality, with a transformation matrix which learns the part-whole relationship between the lower and the higher level capsules.

However, a trainable transformation matrix cannot be defined in practice with variable dimensions. Thus, the dynamic routing in cannot be applied here.
• The key issue with passing the outputs of the PVC layer to the Code Capsule layer is that $N_{pvc}$ is variable with the training sample.

• Prior to routing the lower level capsules to a set of higher level capsules, the lower dimensional capsule outputs need to be projected to the higher dimensionality, with a transformation matrix which learns the part-whole relationship between the lower and the higher level capsules.

• However, a trainable transformation matrix cannot be defined in practice with variable dimensions. Thus, the dynamic routing in cannot be applied here.

Solution : Proposed Variable to Static Routing Algorithm
Algorithm 1 Variable-to-Static Capsule Routing

1: procedure ROUTING(\( \hat{\mathbf{u}}_i, r, a, b \))
2: \( \hat{\mathbf{U}}_{\text{sorted}} \leftarrow \text{sort}( [\hat{\mathbf{u}}_1, ..., \hat{\mathbf{u}}_{N_{pvc}}] ) \)
3: Initialize \( \mathbf{v}_j : \forall i, j \leq a, \mathbf{v}_j \leftarrow \hat{\mathbf{U}}_{\text{sorted}}[i] \)
4: Initialize \( \alpha_{i,j} : \forall j \in [1, a], \forall i \in [1, b], \alpha_{i,j} \leftarrow 0 \)
5: for \( r \) iterations do
6: \( \forall j \in [1, a], \forall i \in [1, b], f_{ij} \leftarrow \hat{\mathbf{u}}_i \cdot \mathbf{v}_j \)
7: \( \forall j \in [1, a], \forall i \in [1, b], \alpha_{i,j} \leftarrow \alpha_{i,j} + f_{ij} \)
8: \( \forall i \in [1, b], \beta_i \leftarrow \text{Softmax}(\alpha_i) \)
9: \( \forall j \in [1, a], \mathbf{s}_j \leftarrow \sum_i \beta_{ij} \hat{\mathbf{u}}_i \)
10: \( \forall j \in [1, a], \mathbf{v}_j \leftarrow \text{Squash}(\mathbf{s}_j) \)
11: return \( \mathbf{v}_j \)
• Often, source code consists of non-essential entities, and only a portion of all entities determine the code class.
• Often, source code consists of non-essential entities, and only a portion of all entities determine the code class.

• $\|\text{Capsule output}\|_2 \propto \text{Prob. of existence}$. 
Often, source code consists of non-essential entities, and only a portion of all entities determine the code class.

\[ \| \text{Capsule output} \|_2 \propto \text{Prob. of existence}. \]

Dependency relationships may exist among entities that are not spatially co-located.
Often, source code consists of non-essential entities, and only a portion of all entities determine the code class.

\[ \text{Capsule output} \|_2 \propto \text{Prob. of existence}. \]

Dependency relationships may exist among entities that are not spatially co-located.

Routing by agreement \( \uparrow \cdot \uparrow = (+) \), \( \uparrow \cdot \rightarrow = (0) \), \( \uparrow \cdot \downarrow = (-) \).
- Code Capsule layer is the final layer of the TreeCaps network, which acts as the classification capsule layer.
• Code Capsule layer is the final layer of the TreeCaps network, which 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.
• For every code capsule $\mu$, the margin loss $L_\mu$ is defined as follows,

$$L_\mu = T_\mu \max(0, m^+ - \|v_\mu\|)^2 + \lambda(1 - T_\mu) \max(0, \|v_\mu\| - m^-)^2$$

(2)

• $T_\mu$ is 1 if the correct class is $\mu$ and zero otherwise.

• $\lambda$ is set to 0.5 to control the initial learning from shrinking the length of the output vectors of all the code capsules.

• $m^+, m^-$ are set to 0.9, 0.1 as the lower bound for the correct class and the upper bound for the incorrect class respectively.
Presentation Outline

1. Introduction to Capsule Networks
2. Methodology
3. Task: Program Classification
4. Limitations and Future Work
Datasets

- **Dataset A**: Python 6 classes of sorting algorithms, with 346 training programs on average per class.
- **Dataset B**: Java 10 classes of sorting algorithms, with 64 training programs on average per class.
- **Dataset C**: C 104 classes, with 375 training programs on average per class.
Quantitative Results

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

<table>
<thead>
<tr>
<th>Model</th>
<th>Dataset A</th>
<th>Dataset B</th>
<th>Dataset C</th>
</tr>
</thead>
<tbody>
<tr>
<td>GGNN</td>
<td>-</td>
<td>85.00%</td>
<td>86.52%</td>
</tr>
<tr>
<td>TBCNN</td>
<td>99.30%</td>
<td>75.00%</td>
<td>79.40%</td>
</tr>
<tr>
<td>TreeCaps</td>
<td>100.00 ± 0.00%</td>
<td>92.11 ± 0.90%</td>
<td>87.95 ± 0.23%</td>
</tr>
<tr>
<td>TreeCaps (3-ens.)</td>
<td>100.00%</td>
<td>94.08%</td>
<td>89.41%</td>
</tr>
</tbody>
</table>

We followed the techniques proposed in Allamanis et al. and BUI et al. to re-generate the results for GGNN and the techniques proposed in Mou et al. to re-generate the results for TBCNN.

- Why Mou et al. reports a higher performance for Dataset C than us?
  - Custom-trained initial embeddings
  - A small set of AST node types defined specifically for C language only

For a fairer comparison (B & C), we used general embeddings based on srcML node vocabulary as the initial embeddings across all models.
Quantitative Results

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

<table>
<thead>
<tr>
<th>Model</th>
<th>Dataset A</th>
<th>Dataset B</th>
<th>Dataset C</th>
</tr>
</thead>
<tbody>
<tr>
<td>GGNN</td>
<td>-</td>
<td>85.00%</td>
<td>86.52%</td>
</tr>
<tr>
<td>TBCNN</td>
<td>99.30%</td>
<td>75.00%</td>
<td>79.40%</td>
</tr>
<tr>
<td>TreeCaps</td>
<td>100.00 ± 0.00%</td>
<td>92.11 ± 0.90%</td>
<td>87.95 ± 0.23%</td>
</tr>
<tr>
<td>TreeCaps (3-ens.)</td>
<td>100.00%</td>
<td>94.08%</td>
<td>89.41%</td>
</tr>
</tbody>
</table>

- We followed the techniques proposed in Allamanis et al. and BUI et al. to re-generate the results for GGNN and the techniques proposed in Mou et al. to re-generate the results for TBCNN.
Quantitative Results

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

<table>
<thead>
<tr>
<th>Model</th>
<th>Dataset A</th>
<th>Dataset B</th>
<th>Dataset C</th>
</tr>
</thead>
<tbody>
<tr>
<td>GGNN</td>
<td>-</td>
<td>85.00%</td>
<td>86.52%</td>
</tr>
<tr>
<td>TBCNN</td>
<td>99.30%</td>
<td>75.00%</td>
<td>79.40%</td>
</tr>
<tr>
<td>TreeCaps</td>
<td>100.00 ± 0.00%</td>
<td>92.11 ± 0.90%</td>
<td>87.95 ± 0.23%</td>
</tr>
<tr>
<td>TreeCaps (3-ens.)</td>
<td><strong>100.00%</strong></td>
<td><strong>94.08%</strong></td>
<td><strong>89.41%</strong></td>
</tr>
</tbody>
</table>

- We followed the techniques proposed in Allamanis et al. and BUI et al. to re-generate the results for GGNN and the techniques proposed in Mou et al. to re-generate the results for TBCNN.

- Why Mou et al. reports a higher performance for Dataset C than us?
  - ✓ Custom-trained initial embeddings
  - ✓ A small set of AST node types defined specifically for C language only
Quantitative Results

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

<table>
<thead>
<tr>
<th>Model</th>
<th>Dataset A</th>
<th>Dataset B</th>
<th>Dataset C</th>
</tr>
</thead>
<tbody>
<tr>
<td>GGNN</td>
<td>-</td>
<td>85.00%</td>
<td>86.52%</td>
</tr>
<tr>
<td>TBCNN</td>
<td>99.30%</td>
<td>75.00%</td>
<td>79.40%</td>
</tr>
<tr>
<td>TreeCaps</td>
<td>100.00 ± 0.00%</td>
<td>92.11 ± 0.90%</td>
<td>87.95 ± 0.23%</td>
</tr>
<tr>
<td>TreeCaps (3-ens.)</td>
<td>100.00%</td>
<td>94.08%</td>
<td>89.41%</td>
</tr>
</tbody>
</table>

- We followed the techniques proposed in Allamanis et al. and BUI et al. to re-generate the results for GGNN and the techniques proposed in Mou et al. to re-generate the results for TBCNN.

- Why Mou et al. reports a higher performance for Dataset C than us?
  ✓ Custom-trained initial embeddings
  ✓ A small set of AST node types defined specifically for C language only

- For a fairer comparison (B & C), we used general embeddings based on srcML node vocabulary as the initial embeddings across all models.
### Model Variant

<table>
<thead>
<tr>
<th>Model Variant</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable-to-Static Routing Algorithm → Dynamic Pooling</td>
<td>83.43%</td>
</tr>
<tr>
<td>Instantiation parameters → $D_{cc} = 4$</td>
<td>90.90%</td>
</tr>
<tr>
<td>$D_{cc} = 8$</td>
<td>92.10%</td>
</tr>
<tr>
<td>$D_{cc} = 12$</td>
<td>90.33%</td>
</tr>
<tr>
<td>$D_{cc} = 16$</td>
<td>91.51%</td>
</tr>
<tr>
<td>TreeCaps → TreeCaps + Secondary Capsule Layer</td>
<td>92.31%</td>
</tr>
<tr>
<td>TreeCaps with Variable-to-Static Routing and $D_{cc} = 8$</td>
<td>92.11%</td>
</tr>
</tbody>
</table>

- Dynamic max pooling is **bad** for capsule networks, as it destroys spatial/dependency relationships.
## Model Analysis

<table>
<thead>
<tr>
<th>Model Variant</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable-to-Static Routing Algorithm → Dynamic Pooling</td>
<td>83.43%</td>
</tr>
<tr>
<td>Instantiation parameters → $D_{cc} = 4$</td>
<td>90.90%</td>
</tr>
<tr>
<td>$D_{cc} = 8$</td>
<td>92.10%</td>
</tr>
<tr>
<td>$D_{cc} = 12$</td>
<td>90.33%</td>
</tr>
<tr>
<td>$D_{cc} = 16$</td>
<td>91.51%</td>
</tr>
<tr>
<td>TreeCaps → TreeCaps + Secondary Capsule Layer</td>
<td>92.31%</td>
</tr>
<tr>
<td>TreeCaps with Variable-to-Static Routing and $D_{cc} = 8$</td>
<td>92.11%</td>
</tr>
</tbody>
</table>

- Dynamic max pooling is **bad** for capsule networks, as it destroys spatial/dependency relationships.

- The instantiation parameters $D_{cc}$ of the CC layer acts as the dimensionality of the latent representation of source code.

  - $D_{cc} \uparrow\uparrow$ - Sparsity and/or correlated instantiation parameters
  - $D_{cc} \downarrow\downarrow$ - Under-representation
1 Introduction to Capsule Networks

2 Methodology

3 Task: Program Classification

4 Limitations and Future Work
Limitations

- Limitations inherited from capsule networks
  - High computational complexity in comparison to CNNs.
  - Relative performance reduction with the increasing number of classes.
Limitations

- Limitations inherited from capsule networks
  - High computational complexity in comparison to CNNs.
  - Relative performance reduction with the increasing number of classes.

- TreeCaps lacks a decoder network, due to which
  - We lose a lot of interpretability.
  - We cannot study the relationship between the learnt instantiation parameters and the physical attributes of data.
Future Work

• Investigate the extent to which TreeCaps can actually capture the dependency relationships of ASTs. How?
Future Work

• Investigate the extent to which TreeCaps can actually capture the dependency relationships of ASTs. How?
  ✓ Integrate a back-tracking mechanism after a forward pass with the test case.
Future Work

- Investigate the extent to which TreeCaps can actually capture the dependency relationships of ASTs. **How?**
  - ✓ Integrate a back-tracking mechanism after a forward pass with the test case.
  - ✓ For a given primary static capsule, the primary variable capsules connected to it with $\phi$-highest coupling coefficients are considered to have dependency relationships.

- Evaluate the effectiveness of TreeCaps as an embedding generating technique.
- Extend TreeCaps to other related tasks such as bug detection and localization.
Future Work

- Investigate the extent to which TreeCaps can actually capture the dependency relationships of ASTs. How?
  - ✓ Integrate a back-tracking mechanism after a forward pass with the test case.
  - ✓ For a given primary static capsule, the primary variable capsules connected to it with $\phi$-highest coupling coefficients are considered to have dependency relationships.
  - ✓ We subsequently compare related pieces of code identified by TreeCaps to program dependencies identified by program analysis techniques.

- Evaluate the effectiveness of TreeCaps as an embedding generating technique.
- Extend TreeCaps to other related tasks such as bug detection and localization.
Future Work

• Investigate the extent to which TreeCaps can actually capture the dependency relationships of ASTs. How?
  ✓ Integrate a back-tracking mechanism after a forward pass with the test case.
  ✓ For a given primary static capsule, the primary variable capsules connected to it with $\phi$-highest coupling coefficients are considered to have dependency relationships.
  ✓ We subsequently compare related pieces of code identified by TreeCaps to program dependencies identified by program analysis techniques.

• Evaluate the effectiveness of TreeCaps as an embedding generating technique.
Future Work

• Investigate the extent to which TreeCaps can actually capture the dependency relationships of ASTs. How?
  ✓ Integrate a back-tracking mechanism after a forward pass with the test case.
  ✓ For a given primary static capsule, the primary variable capsules connected to it with $\phi$-highest coupling coefficients are considered to have dependency relationships.
  ✓ We subsequently compare related pieces of code identified by TreeCaps to program dependencies identified by program analysis techniques.

• Evaluate the effectiveness of TreeCaps as an embedding generating technique.

• Extend TreeCaps to other related tasks such as bug detection and localization.
This work was accepted to be presented at NeurIPS workshops this year!

Thank you!