Tree-to-tree Neural Networks for Program Translation

Abstract

Program translation is an important tool to migrate legacy code in one language into an ecosystem built in a different language. In this work, we are the first to employ deep neural networks toward tackling this problem. We observe that program translation is a modular procedure, in which a sub-tree of the source tree is translated into the corresponding target sub-tree at each step. To capture this intuition, we design a tree-to-tree neural network to translate a source tree into a target one. Meanwhile, we develop an attention mechanism for the tree-to-tree model, so that when the decoder expands one non-terminal in the target tree, the attention mechanism locates the corresponding sub-tree in the source tree to guide the expansion of the decoder. We evaluate the program translation capability of our tree-to-tree model against several state-of-the-art approaches. Compared against other neural translation models, we observe that our approach is consistently better than the baselines with a margin of up to 15 points. Further, our approach can improve the previous state-of-the-art program translation approaches by a margin of 20 points on the translation of real-world projects

My notes

Nowadays, to translate programs between different programming languages, typically programmers would manually investigate the correspondence between the grammars of the two languages, then develop a rule-based translator. However, this process can be inefficient and error-prone. In this work, we make the first attempt to examine whether we can leverage deep neural networks to build a program translator automatically.

In this work, we observe that the main issue of an RNN that makes it hard to produce syntactically correct programs is that it entangles two sub-tasks together:

(1) ***learning the grammar;*** and (2) ***aligning the sequence with the grammar*.** When these two tasks can be handled separately, the performance can typically boost. For example, Dong et al. employ a tree-based decoder to separate the two tasks. In particular, the decoder in leverages the tree structural information to (1) **generate the nodes at the same depth of the parse tree using an LSTM decoder;** and (2) ***expand a non-terminal and generate its children in the parse tree*.** Such an approach has been demonstrated to achieve the state-of-the-art results on several semantic parsing tasks.

(Program translation). Given two programming languages L_s and L_t each being a set of instances (p_k, T_k), where p_k is a program, and T_k is its corresponding parse tree. We assume that there exists a translation oracle PI, which maps instances in L_s to instances in $L_{t.}$ Given a dataset of instance pairs (i_s ,i_t) such that i_s belongs to L_s, i_t belongs to L_t and $\pi(i_s) = i_t$,our problem is to learn a function F that maps each i_s belongs to L_s into $i_t = \pi(i_s)$.

They design the **tree-to-tree neural network**, which follows an **encoder-decoder framework** to encode the source tree into an embedding, and decode the embedding into the target tree. To capture the intuition of the **modular translation process**, the decoder employs an **attention mechanism** to locate the corresponding source sub-tree when expanding the non-terminal.

We evaluate our tree-to-tree model against a **sequence-to-sequence model**, a **sequence-to-tree model**, and a **tree-to-sequence model**. *Note that for a sequence-to-sequence model, there can be four variants to handle different input-output formats. For example, given a program, we can simply tokenize it into a sequence of tokens. We call this format as raw program, denoted as* **p** *We can also use the parser to parse the program into a* **parse tree**, and then serialize the parse tree as a sequence of tokens. Our serialization of a tree follows its depth-first traversal order, which is the same as. We call this format as parse tree, denoted as **T**. For both input and output formats, we can choose either P or T. For a ***sequence-to-tree model***, we have two variants based on its input format being either P or T; note that the sequence-to-tree model generates a tree as output, and thus requires its output format to be T (unserialized). Similarly, the tree-to-sequence model has two variants, and our tree-to-tree only has one form. Therefore, we have 9 different models in our evaluation.

In this work, they are the first to consider neural network approaches for the program translation problem, and are the first to demonstrate a successful design of ***tree-to-tree neural network combining both a tree-RNN encoder and a tree-RNN decoder*** for translation tasks. Extensive evaluation demonstrates that our tree-to-tree neural network outperforms several state-of-the-art models. This renders our tree-to-tree model as a promising tool toward tackling the program translation problem. In addition, they believe that their proposed tree-to-tree neural network has the potential to generalize to other tree-to-tree tasks, and they consider it as future work.