International Journal of Engineering Applied Sciences and Technology, 2021
Vol. 6, Issue 8, ISSN No. 2455-2143, Pages 191-195
Published Online December 2021 in IJEAST (http://www.ijeast.com)

TRANSLATION OF NATURAL LANGUAGE TO CODE: A SURVEY

Taha Yunus Moochhala,
Student MCA Department
Jain University, Bengaluru, Karnataka, India

Dr. Kamalraj R
Assistant Professor MCA Department
Jain University, Bengaluru, Karnataka, India

***Abstract*- Learning to code in a new language takes a lot of time and effort for new as well as seasoned developers. There is a certain language that machines understand and we humans have to learn that language in order for us to communicate with the machine and make it do the things that we desire. To reduce this human-machine barrier of communication researchers have come up with a machine learning solution to translate natural language into code that allows developers to spend more time on logic and architecture rather than the actual instructions given to the machine. This paper surveys the approaches developed by various research scholars to translate natural language into machine understandable code. We analyze these approaches and state their boons and banes so that there is scope for improvement in this domain. This paper aims to give an idea about the various approaches in this domain to budding researchers wanting to contribute in this field.**

***Keywords*- neural machine translation, python, LSTM, attention, encoder, decoder, tokenizers, transformers**

I. INTRODUCTION

There are approximately 700 different programming languages present in the world. Learning a programing language takes a lot of time and effort. In this current era developers are required to keep up with the pace of the ever-growing industry which has become increasingly difficult. Shorter deadlines and faster results have become a common practice in the industry. The problem does not only impact the developers, in the bigger picture it also effect companies by delays in the project and less efficient development.

With the help of Machine Learning and Natural Language Processing, Researchers and developers have come up with a solution in which instead of giving instructions to the machine in high level computer language the user would give instructions in normal spoken English which would then be translated into machine understandable code. Availability of large scale dataset generated by many repositories, innovation and improvement of machine learning algorithms, and rapid development in computing capacity have all contributed to the development of such projects.

Machine translation (MT) has become a commonly used tool in all aspects of life. Linguists, sociologists, computer scientists and even the common man take benefit of this application, by processing natural language to translate it into some other natural language [1]. There are different approaches to machine translation the most common one being statistical machine translation and neural machine translation used widely in the current era.

Statistical machine translation uses probability to translate from one language to another. In this approach the model assumes that every source S has a possible translation T and assigns a probability P(T|S) for every (S,T) sentence pair. [2]. A word sequence that is in testing is likely to be different from all the word sequences during training, this problem cannot be addressed by a statistical approach one major reason being the curse of dimensionality.

Neural machine translation uses a deep neural network approach to translate sequences from one language to another. Neural machine translation or the sequence to sequence (seq2seq) model uses many different approaches including multiple recurrent neural networks [3] attention mechanism [4] and the current state of the art transformer architecture.

II. RELATED WORK

During the survey we found many different approaches taken for this application. The approaches that used neural machine translation provided the best results. Hence we decided to stick with this approach and conduct our survey on the basis of these findings.

The following sub sections are organised as follows. Section A and Section B provides an insight about neural machine translation using LSTMs. Section C and Section D provides details about using Transformers to solve translation of natural language to code

A. A Syntactic Neural Model for General-Purpose Code Generation

During the survey we found that there are many approaches where researchers used LSTMs with an attention mechanism to translate natural language to code. This approach outperformed previous semantic parsing approaches

In this paper, given a natural language intent x which is to be converted to a code snippet c, problem is taken head on by first generating an underlying AST. The generation of AST is done by a grammar model that splits this process into a sequence of tree-constructing actions. This procedure actually starts with an initial derivation AST with a single root node. The generation process then follows a depth-first left-to right order choosing the APPLYRULE action following a close-set of grammar rules to form the general structure of the tree. The GENTOKEN action is called on the leaf nodes.

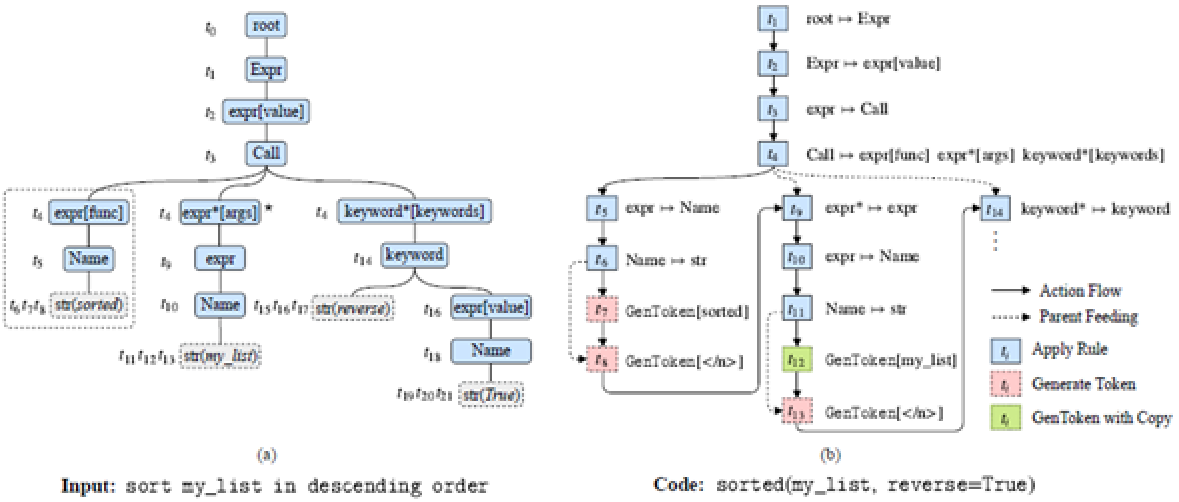


Figure 1:(a) the Abstract Syntax Tree (AST) . (b) the action sequence (up to t14) used to generate the AST in (a)

Before we see what these actions are let us understand a bit about what an AST is. An AST or an abstract Syntax tree is a representation of source code or in other words a set of production rules containing a head node and multiple child nodes. [5]

The APPLYRULE chooses a rule r from a set of rules it uses the rule to expand a specific node by appending all its child nodes as shown in figure 1 at time step t4 taken from [5]. The APPLYRULE is basically used to grow the derivation AST.

The GENTOKEN action is called when the program reaches a frontier node corresponding to a variable type. This action is used to fill the node with values irrespective of the number of token the variable has, that is if the variable is storing the name of a function it takes one token on the other hand a variable storing a string has more than one token. The GENTOKEN also terminates the node and the program moves on to the new node. [5]

The probability of generating an AST y is given by the formula

$$p(y|x) = \prod_{t=1}^{T} p(a_t \,|x, a_{<t})$$

The above formula is parameterized using a neural encoder – decoder based LSTM. The encoder is a standard Bi-directional LSTM that encodes the inputs into vectorial representation. The decoder is also a standard LSTM that makes use of hidden states at each timestamp determined by the LSTM transition function. The decoder also makes use of the attention vector to give us the desired output. This method produced BLUE score of 84.5 on the Django dataset.

The advantages of this model are, the problem with LSTM without attention was the fixed length internal representation making the LSTM unable to take long sequences of inputs. This was solved by keeping the intermediate outputs from the encoder LSTM from each step of the input sequence and training the model to learn to pay selective attention to these inputs and connect them to items in the output sequence.

Disadvantages of this model are the increased computational burden on the machine and slower training as there are no parallel computing capabilities.

B. TRANX: A Transition-based Neural Abstract Syntax Parser for Semantic Parsing and Code Generation

In continuation of the previous paper Neubig et al [5] made an addition to the previous model that provides better results as compared to the existing neural model. The authors designed TRANX or a Transition based Abstract Syntax Parser which mainly has two great advantages over the previous model that are, its highly accurate as it uses information of the syntax of

the target meaning representation, to model the information flow in the target language and It is generalizable meaning it can be adapted to almost any language just by generating a new AST description.[6]
As seen before this model also uses a grammar model called a transition system which contains three different actions APPLYCONSTR, GENTOKEN, and REDUCE .

- APPLYCONSTR: This action actually builds the AST by appending Child nodes of similar type following certain rules like sequential cardinality.
- GENTOKEN: Once we reach a node with a variable type (eg: str) which is used to fill the node with values. Each variable/constant can contain values with one or multiple tokens (eg: storing a function name has one token, storing a string may have multiple tokens) To stop the generation of token a special GENTOKEN[</f>] is called.
- REDUCE: action marks the completion of generation of child values.

This model also uses the same probability formula to generate the AST which is parameterised by a neural encoder – decoder based LSTM as seen in the previous paper. The results of the TRANX model over three different kinds of datasets. The first one being GEO and ATIS produced an accuracy of 87.7 and 86.2 respectively. For the Django dataset it produced an accuracy of 73.7 and lastly the WikiSQL dataset produced 78.6 accuracy.

C. Natural Language to Python Source Code using Transformers

The transformer model was proposed by Google to carry out seq2seq tasks well. There are many benefits of the transformer model when compared to other seq2seq models like LSTM .The transformer model allows for more parallelization in GPUs. It can produce state of the art translations after being trained for as little as twelve hours on eight P100 GPUs [4]. Transformers are faster and more robust in comparison to LSTMs [7].
The transformer model used by Meet Shah et al in their paper [7] produced a BLEU score of 64.2. They have beautifully explained the transformer model by splitting it into four main parts.

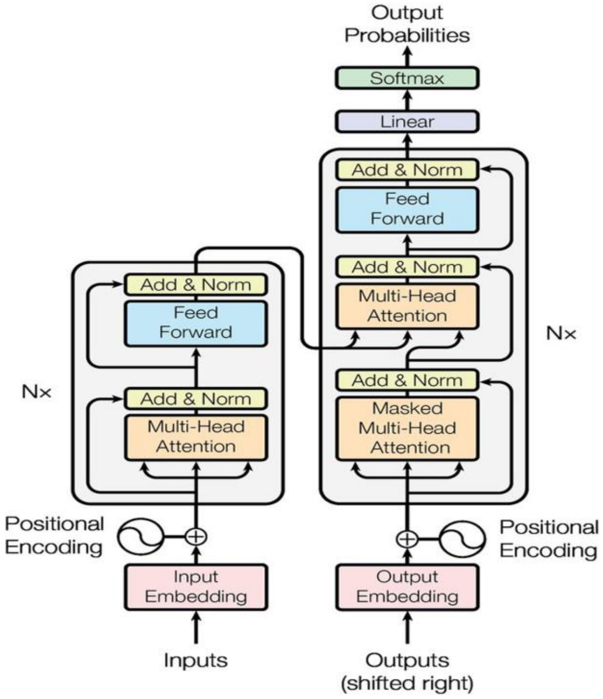


Figure 2: The Transformer Model

The first part being the dataset, here they have used ase15-django-dataset [8] which consists of 18805 records in English and its corresponding python snippets. The second part is the pre-processing of the dataset, here the dataset is split into train and test containing 1500 and 3805 rows respectively. The dataset is then converted into tensorflow dataset format as they are using the tensorflow library to train the model.
Tokenizer comes in next in which they use the bert_vocab_from_dataset library from tensorflow which generates vocabulary for both English and python. It is common knowledge that a machine cannot understand words or sentences it can only understand number so this step is vital to be carried out. The vocabulary size was set to 4000 and two bert tokenizers were trained on English and python, Also once the translation is completed the tokenize0072 is once again used to transform the output back into understandable code in python.
Coming towards what exactly is a transformer model. The transformer model consists of an encoder layer, a decoder layer and a final linear layer. The entire transformer model consists of stacked self-attention and feed-forward fully connected layers for both encoder and decoder as shown in figure 2. [7]
In transformer model the model doesn't have any idea about the position of the word in a sentence because it processes all words simultaneously. Hence, an extra piece of Information telling the details of the position of the word is a must for this model which is carried out by positional encoding. [7]

The encoder first performs the input embedding then does positional encoding and then it sends the data to many encoding layers. As shown in figure 2 the mutlihead attention which makes the encoding layer splits many linear layers then performs the dot product attention and then mergers to form a final linear layer.

As for the decoder, it performs an output embedding then positional encoding, and finally data goes through multiple decoding layers.

Advantages of the model are that it is Non sequential meaning sentences are processed as a whole rather than word by word, therefore larger sequence is taken at a time. It has self-attention which helps the model focus on connections but within the same sentence and lastly positional embedding in which the idea is to use fixed or learned weights which encode information related to a specific position of a token in a sentence (gives position of a word).

Disadvantages of the model include attention which can only deal with fixed-length text strings. The text has to be split into a certain number of segments before being fed into the system as input. Chunking of text causes context fragmentation.

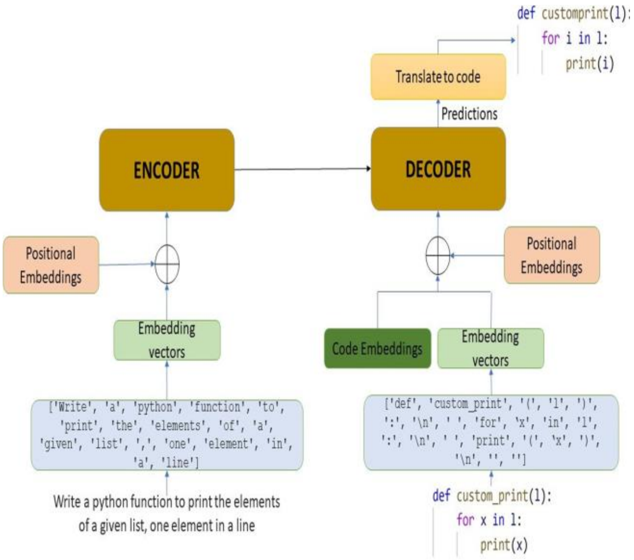
D. Text2PyCode: Machine Translationof Natural Language Intent to Python Source Code

In this approach S. Bonthu et al took the approach taken by Meet Shah et al in their paper [7] a step further by introducing code embedding in their model. They also use a different tokenizer which is the Spacy tokenizer.

In this paper the dataset was extracted from public repositories like github and stackoverflow. Since the dataset was crowd sourced it is very noisy which was later pre-processed and cleaned. Neural machine translation system show lower translation quality on long sequences [9], so all source codes having length higher than 1000 characters were omitted. After the pre-processing they ended up having 4299 rows of data. The dataset was then split into train and test, training set containing 85 % of the dataset.

The model implementation was carried out in three stages so that researchers could compare the results.

First being the baseline model which used the implementation of the Sockeye toolkit [10] containing 3 sub layers and 8 heads for multi-head attention mechanism. Next being the initial model where the Spacy tokenizer was used, here the encoder and decoder layer of the model were also fed with the word embedding created by following the Word2Vec algorithm [11].


Translate to code
Predictions
ENCODER
DECODER
Positional Embeddings
Positional Embeddings
Embedding vectors
Code Embeddings
Embedding vectors
['Write', 'a', 'python', 'function', 'to', 'print', 'the', 'elements', 'of', 'a', 'given', 'list', ',', 'one', 'element', 'in', 'a', 'line']
['def', 'custom_print', '(', 'l', ')', ':', '\n', ' ', 'for', 'x', 'in', 'l', ':', '\n', ' ', 'print', '(', 'x', ')', '\n', '', '']
Write a python function to print the elements of a given list, one element in a line

```
def customprint(l):
    for i in l:
        print(i)
```

```
def custom_print(l):
    for x in l:
        print(x)
```

Figure 3:Architecture of the final model.

In the final model the word embeddings of the source code was generated using the GenSim library on the CoNala python corpus. These pre-trained vectors are also added while building vocabulary for the model. The final model is shown in figure 3 taken from [11]. This model has a BLEU score of 32.40 and a ROUGE score of 85.1 which Is at par with results of natural language processing.

III. CONCLUSION

This paper surveys different methods in translating natural language intent into code. We can conclude that this project will allow us to save time and efforts of a developer by making the while process of development easier. Not only does it save time but also in turn it saves money for the company developing a project by making the development faster and easier. It has tremendous potential in making an impact in this industry and making things easier for the people working in this industry. This project has a lot of scope for improvement and which is being worked on by many research scholars as we speak.

IV. REFERENCES

[1]. A. Garg and M. Agarwal, “Machine Translation: A Literature Review,” arXiv [cs.CL], 2018.

[2]. P. Brown et al., “A statistical approach to machine translation,” in Readings in Machine Translation, The MIT Press, 2003.

[3]. K. Cho et al., “Learning phrase representations using RNN encoder-decoder for statistical machine translation,” arXiv [cs.CL], 2014.

[4]. A. Vaswani et al., “Attention is all you need,” arXiv [cs.CL], 2017.

[5]. P. Yin and G. Neubig, "A syntactic neural model for general-purpose code generation," arXiv [cs.CL], 2017.

[6]. P. Yin and G. Neubig, "TRANX: A transition-based neural abstract syntax parser for semantic parsing and code generation," in Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, 2018.

[7]. M. Shah, R. Shenoy, and R. Shankarmani, "Natural language to python source code using transformers," in 2021 International Conference on Intelligent Technologies (CONIT), 2021, pp. 1–4.

[8]. Y. Oda et al., "Learning to generate pseudo-code from source code using statistical machine translation," in 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), 2015, pp. 574–584.

[9]. P. Koehn and R. Knowles, "Six Challenges for Neural Machine Translation," arXiv [cs.CL], 2017.

[10]. F. Hieber et al., "Sockeye: A Toolkit for Neural Machine Translation," arXiv [cs.CL], 2017.

[11]. S. Bonthu, S. R. Sree, and M. H. M. Krishna Prasad, "Text2PyCode: Machine translation of natural language intent to python source code," in Lecture Notes in Computer Science, Cham: Springer International Publishing, 2021, pp. 51–60.