NL2code

Sai Krishna Karanam karanam.s@husky.neu.edu
Nischal Mahaveer Chand mahaveerchand.n@husky.neu.edu
Varun Sundar Rabindranath rabindranath.v@husky.neu.edu

I. Introduction

A. Importance of NL2code

Recent Integrated Development Environments (IDEs) provide a feature called "context-aware code completion" which helps programmers speed up the coding process. This is very helpful for programmers who are already aware of the programming language syntax and the design paradigms. But, for beginners, this can be time consuming.

The first step, for beginners, is to search how to implement the code in natural language XXX. Due to the vast amount of resources available online, may unreliable or outdated, having a local NL to code generator will make programming more enjoyable for beginners and more productive for veteran programmers. And with the introduction of a new language almost everyday, it has become more and more important to establish an interface between NL and code.

We aim to create neural based models to generate NL to code. Below we describe various attempts of implemening various encoder-decoder models for the same. It should be noted that the project does not aim at providing a framework to build industry standard deployable code, but is rather a proof-of-concept. The scope of this project is limited to generate an accurate single line program statement given user intent in the form of a comment or description. XXX

Examples

input	call the function sorted with argument x
ref.	sorted(x)
input	for every k in keys
ref.	for k in keys:

II. Related Word

Techniques to generate regular expressions [1], input parsers [2], UML diagrams, object oriented class layout, and general purpose code from Natural Language (NL) specification have been researched for a long time.

Recent approaches in converting NL to executable source code predominantly use neural network techniques. These approaches follow two basic patterns. One, is to directly convert NL to source code by modeling the task as a Sequence-to-Sequence conversion task, using Recurrent Neural Networks (RNN) (Lili Mou et al., 2015, Ling et al., 2016) [3], [6] to that end. The other is to generate Abstract Syntax Trees (ASTs) from NL input, the deterministically convert the produced AST to code (Pengcheng Yin et al., 2017) [4].

However, research [4] shows that the second approach is better than the first and emphasizes the use of syntax information of the target language, in NL to code tasks. We consider the recent research paper by Pengchen Yin et. al. [4] , on syntactic neural model for code generation as the primary motivation for this project. [4] proposes an approach where they translate NL to an Abstract Syntax Tree (AST) first, thus using the grammar of the target language as a prior knowledge. They report a 10% absolute improvement in accuracy compared to the previous state-of-the-art on standard datasets. Since, converting from AST to code can be done deterministically, their system always produces syntactically correct, executable code. This aspect is lacking in the Sequence-to-Sequence models, where the correctness of the generated code is not guaranteed. They use a Bidirectional LSTM (BiLSTM) network for encoding each word of the NL input as a context specific embedding, and an RNN as a decoder to generate an output AST.

Maxim Robinovich et. al. [5], propose an approach similar to [4] for converting NL to AST. They confirm the experimental results of [4], thus again emphasizing the use of target language syntax. However Maxim Robinovich et. al., have evaluated their system on a wider range of datasets achieving near state-of-the-art results on average.

Pengcheg Yin et.al [4], report that 25% of the errors incurred by their system, on one of the datasets, is due to the generated code only partially implementing the required functionality. We hypothesize that this could be due to the standard datasets used for this task being noisy. We propose a denoising procedure, explained in the Data Preprocessing section XXX to clean the dataset, thereby making the datasets more generic.

Barone et. al, 2017, [7], hereafter referred to as the "Edinburgh paper", explores the difficulty of the code generation task and critiques the currently available datasets for this task. They also comment that the effectiveness reported by [4] should not be taken at face value.

In the Edinburgh paper, Barone et al. scraped online Opensource Python repositories to extract "docstring-code" pairs. They propose that such data accurately represents the query/code pairs found in production environment. They used a Neural Machine Translation approach to translate docstring to code and reported a baseline BLEU score of 10.24, While [4] reports a BLEU score of 84.5 on the Django dataset.

We wanted to check how [4] performs on a similar dataset. We scraped query-code pairs from the same online Opensource Python repositories as [7]. [4], on our scraped dataset reports a BLEU score of only 14.2 which reaffirms the [7]'s claim that the reported effectiveness of various models could not be taken at face value and that the task of code generation is more difficult than what is conceived by the community. Much of this misrepresentation could be attributed to the fact that the datasets currently available for this task is very limited and noisy.

III. Dataset

A. Standard Datasets

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

1) Django: Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

2) HS: Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

B. Our dataset

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

IV. Models

We describe four models, each is build upon Pengcheng's model, but encodes the input in a different method. First we start by briefly describing Pengcheng's model.

A. Pengcheng's model

Pengcheng recognized that adding syntax information to the model would give better predictions results XXX. Their model follows an encoder-decoder architecture, and takes the raw comment as input and generates an AST of the corresponding code as output.

The encoder comprises of an embedding layer and a BiLSTM layer. It takes a comment as input, embeds each word in the comment to give token embeddings TE_t, for each word t in the comment. Each TE_t is sequentially feed into the BiLSTM layer, to produces a Query Embedding (QE) of 128 dimensions. QE is passed to the decoder module.

The decoder uses an Recurrent Neural Network (RNN) to sequentially generate each node of the AST. Each node maps to a timestep in the RNN decoding process and thus, generating the AST can be interpreted as unrolling the RNN. The RNN also maintains an internal state to track the generation process at each timestep.

Pengcheng tackles the problem with a probabilistic grammar model of generating an AST y given NL description x: $p(y|x)$. The best possible AST $\hat{y}$ is given by:

$$\hat{y} = \arg\max_{y} p(y|x) \tag{1}$$

$\hat{y}$ is then deterministically converted to the corresponding Python code using astor XXX.

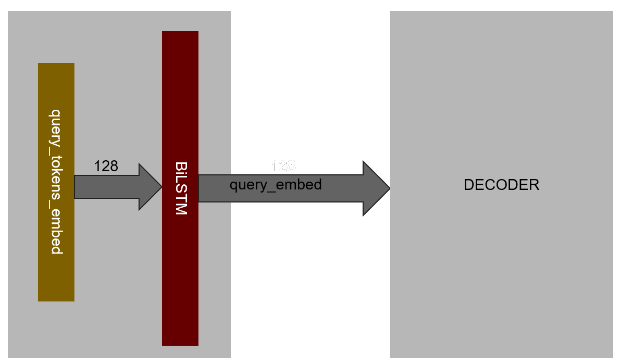

Figure 1: Architecture of Pengcheng's model. The encoder embeds each token of the NL input to give TE_t, which is then passed into the BiLSTM layer. The encoder produces a 128 dimension QE which is feed into the decoder.

B. Our models

As described, the decoder is already state-of-the-art XXX, and needs no further modifications. All models described hereon use the same decoder architecture as Pengcheng's model with modifications only to the encoder and the input data XXX. All models are trained and tested using the dataset decribed in SECTION XXX.

1) Basic Concat (BC): For our first attempt to incorporate syntax information into the encoder, we decided to concatenate (shown as ":") the POS ID and phrase ID of each token to the corresponding token embedding, giving the Augmented Token Embedding (ATE). The ATE is then feed into a modified BiLSTM layer that takes 130 dimension embeddings, rather than the default 128 dimensions.

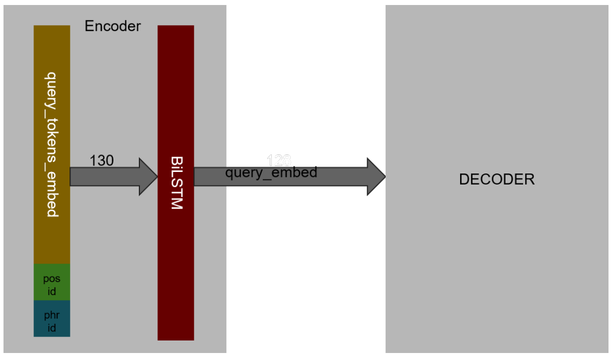

Figure 2: Architecture of Basic Concat model. Notice that the POS ID and phrase ID are directly appended to the corresponding TE. The resulting 130 dimension embedding is passed to the BiLSTM.

$\mathrm{TE_t}$ dimension: 128
$\mathrm{POS\_ID_t}$ and $\mathrm{Phrase\_ID_t}$ dimension: 1 each; total 2
$\mathrm{ATE_t}$ dimension = 130

$\mathrm{ATE_t} = [\mathrm{TE_t} : \mathrm{POS\_ID_t} : \mathrm{Phrase\_ID_t}]$
$\mathrm{QE} = \mathrm{BiLSTM(ATE)}$

2) Linear Projection (LP): To add syntactic information over a sequence of tokens, we used an embedding layer for POS and phrase tags. The resulting TE, POS embedding (POSE), and Phrase embedding (PhE) are then concatenated to produce the ATE; which is a $(128 * 3)$ dimension vector. We then apply a linear projection (using a dense layer with the linear activation function) to give ATE Projected (ATEP) which is passed to BiLSTM.

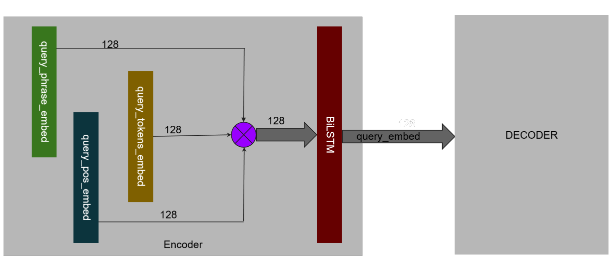

Figure 3: Architecture of Linear Projection model. Each embedding - TE, POSE, and PhE are concatenated and projected (shown as purple circles). The resulting ATEP is passed to the BiLSTM.

$\mathrm{TE_t}$ dimension: 128
$\mathrm{POSE_t}$ dimension: 128
$\mathrm{PhE_t}$ dimension: 128
Dense = (128 * 3) input nodes, 128 output nodes

$\mathrm{ATE_t} = [\mathrm{TE_t} : \mathrm{POSE_t} : \mathrm{PhE_t}]$
$\mathrm{ATEP_t} = \mathrm{Dense}(\mathrm{ATE_t})$
$\mathrm{QE} = \mathrm{BiLSTM(ATEP)}$

3) Linear Projection Reduced Dimension (LPrd): Subsequently, we noticed that the POS and Phrase vocabulary sizes were relatively smaller than token vocabulary size. To avoid redundancy XXX, we reduce the embedding dimensions of POSE and PhE to 8 and 32 respectively. The process described in LP is then repeated.

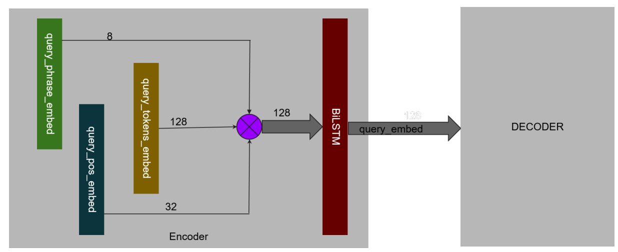

Figure 4: Architecture of Linear Projection Reduced Dimension model. Same as LP, but with reduced dimensions for POSE and PhE.

$\mathrm{TE_t}$ dimension: 128
$\mathrm{POSE_t}$ dimension: 8
$\mathrm{PhE_t}$ dimension: 32
Dense = (128 + 8 + 32) input nodes, 128 output nodes

$\mathrm{ATE_t} = [\mathrm{TE_t} : \mathrm{POSE_t} : \mathrm{PhE_t}]$
$\mathrm{ATEP_t} = \mathrm{Dense}(\mathrm{ATE_t})$
$\mathrm{QE} = \mathrm{BiLSTM(ATEP)}$

4) Raw Query Independent Preprojection (AdvLP): Rather than applying one linear projection on ATE, we apply two here, where the first is independent of the input query. POSE and PhE are concatenated and projected to create Augmentation Embedding (AE), which is then concatenated with TE and projected to produce QE. We hypothesize that this would preserve most of the information from the input query during projection.

$\mathrm{TE_t}$ dimension: 128
$\mathrm{POSE_t}$ dimension: 128
$\mathrm{PhE_t}$ dimension: 128
Dense = (128 * 2) input nodes, 128 output nodes

$\mathrm{AE_t} = \mathrm{Dense}([\mathrm{POSE_t} : \mathrm{PhE_t}])$
$\mathrm{ATE_t} = [\mathrm{TE_t} : \mathrm{AE_t}]$
$\mathrm{ATEP_t} = \mathrm{Dense}(\mathrm{ATE_t})$
$\mathrm{QE} = \mathrm{BiLSTM(ATEP)}$

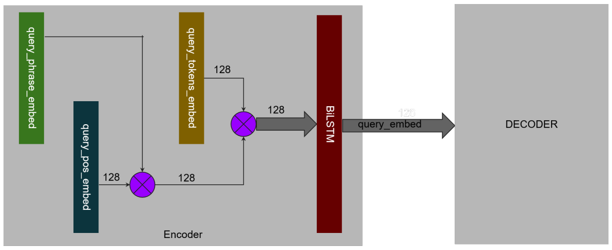

Figure 5: Architecture of Raw Query Independent Pre-projection model. POSE and PhE are concatenated and projected independent of the input query. The resulting is concatenated with TE and projected to produce QE.

V. Experiments

All decoder dimensions and configurations are left untouched and are thus the same as in Pengcheng's model. For each of the above models, the embedding sizes are as described Models. Each model is run for a maximum of 50 epochs, with early stopping if the validation metrics do not change for 10 epochs.

VI. Results

Models		Metrics	
		BLUE	Accu.
Pengcheng		84.5	71.6
Base		73.2	67.9
NL2code	BC	73.6	69.4
	LP	74.3	69.7
	LPrd	73.6	69.0
	AdvLP	73.7	69.1

VII. Conclusion

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.