Analysing a Natural Language to Code Translation System Applied to Spanish Queries

Nitin Balachandran
nitinkb2015@gmail.com

Yunheng Han
yhhan@terpmail.umd.edu

Zeping He
zepinghe@umd.edu

John Kastner
kastner@umd.edu

Callie Y Kim
ckim1123@terpmail.umd.edu

Anjali Mittu
amittu@umd.edu

Suteerth Vishnu
svishnu@terpmail.umd.edu
University of Maryland
College Park, MD

Abstract

Generating code from natural language is an important task, as often times, people are able to verbally state their intention but are unable to translate this intent into code. With the advent of machine learning, models are able to convert natural language into code. However, these approaches only address this challenge in English and not in other languages. In this paper we explore how current methods of code generation from natural language perform when generating code from natural languages other than English, specifically Spanish. While our experiments show lower BLEU score on Spanish datasets than English datasets, we do not believe we can conclude that the evaluated model generally performs worse on Spanish than English due to concerns about the quality and quantity of our Spanish data.

1 Introduction

The goal of our project is to investigate whether an existing natural language to code conversion technique generalizes well to languages other than English, specifically Spanish. It is often easier to describe code in Natural Language than to immediately know the programming language specific implementation. This has lead to the rise of Stack Overflow, an open online community where users can ask for help with their programming problems. Questions on Stack Overflow are often phrased so that a natural language intent is given in the title. An answer will then include code which implements the intent. Stack Overflow is helpful to new and veteran software engineers. More than 52 million people visit Stack Overflow each month to post new questions, answer questions, or look at more than 18 million answered questions (Stack Exchange Inc, 2019). Natural Language to code generators could help speed up the process of looking up questions by allowing users to directly generate code instead.

Techniques for natural language to code conversion have been developed over the past decades. Early work required controlled inputs with restricted syntax. However, modern approaches can reach high levels of accuracy in producing unbound output. Code generation is difficult because the output must be structured. It is hard for sequence-to-sequence models to output code that is well-formed. To solve this problem, ASTs were used as output from the sequence-to-sequence models (Rabinovich et al., 2017). Code generation has been demonstrated with various output languages including lambda calculus (Yin and Neubig, 2018), SQL (Yin and Neubig, 2018), Java (Iyer et al., 2018), and Python (Yin and Neubig, 2017; Rabinovich et al., 2017).

In this paper, we focused on the translation techniques from NL2Code (Yin and Neubig, 2017) and TranX (Yin and Neubig, 2018) to produce Python code. The original NL2Code paper uses three different datasets: HearthStone, Django and IFTTT. Instead, we focused on using the CoNaLa dataset. It is interesting to see how existing approaches generalize to natural languages other than English. It is important for a model to be able to learn from languages other than English, since code is not just written by English speakers. We compare the performance of the TranX translation techniques on

English and Spanish using the CoNaLa dataset, the CoNaLa dataset translated into Spanish and a Spanish dataset mined from the Spanish language Stack Overflow website[1].

2 Related Work

2.1 Early Methods

Natural language descriptions are ambiguous and imprecise, which lead to difficulty in generating executable code. As a result, controlled natural language, a subset of language with restricted syntax and semantics, is used in early work (Schwitter and Fuchs, 1996). The input text, if controlled, can be parsed easily through rule-based approaches. This method is able to generate executable code effectively on the basis of controlled language. However, natural language descriptions from users are usually unrestricted. This method therefore has inevitable limitations in practice.

In Wong and Mooney (2006) a statistical approach to learn semantic parsing from sentences was proposed. It consists of a synchronous context-free grammar (SCFG) and a probabilistic model on possible derivations. The learning task is twofold: the rule induction and the probabilistic model induction. To induce the rules, the authors train a word alignment model and extract the transformation rules from most probable word alignments between the sentences and the mean representations (MR). For the probabilistic model, the authors use a maximum entropy model of conditional probability over the derivations and observations.

The approach from Wong and Mooney (2006) is able to translate a natural language sentence into the MR, which is in the form of a nested structure. The nested structure then can be converted into executable code, so it is useful in code generation tasks. Nevertheless, the approach is based on statistical models, while in this project, we utilise a framework taking advantage of learning methods, and expect better performance.

2.2 Reinforcement Learning

Machine learning methods are widely used in modern approaches to problems in Natural Language Processing, and achieve state-of-the-art results in many tasks. In Branavan et al. (2009), the authors proposed a reinforcement learning framework to generate executable actions from natural language instructions in the Microsoft Windows troubleshooting guide. The authors first train a model with a simple reward information, whose performance is close to the fully supervised model. Afterwards, the performance is further improved by data augmentation.

Reinforcement learning has been proven to be effective in action generation. Note that the instructions are given in a sequential order, and the actions are generated sequentially as well so, a different frame work is required to generate code with nested structures.

[1]https://es.stackoverflow.com

2.3 Encoder-Decoder Models

In order to better generate code with nested structures, Yin and Neubig (2017) introduced the combined use of a neural network model along with a grammar model. The authors used an encoder-decoder model based on a bidirectional long short-term memory (BiLSTM), which generates an Abstract Syntax Tree (AST). The decoder uses a grammar model, which is used to create the AST from a sequence of tree-constructing actions. There are three possible actions at each step: `APPLYCONSTR`, `REDUCE`, and `GENTOKEN`. The set of actions and terminals are used by the decoder to produce the AST. Yin and Neubig (2017) showed improvements of 11.7% and 9.3% over previous work. The models created by the authors were tested over three datasets: HearthStone, Django, and IFTTT.

TranX (Yin and Neubig, 2018) expanded on this work by introducing the use of parent feeding in the decoding stage. The parent feeding vector contains information about the parent's embedding and the state of its constructor. TranX produced a 1% improvement in accuracy over Yin and Neubig (2017) when applied to its Python datasets.

Iyer et al. (2018) expanded on the neural network grammar model from Yin and Neubig (2017). Related to our work, the decoder they chosen is an an LSTM-based RNN that produces a context vector at each step. Each vector is used to compute a distribution over next possible actions from the grammar model. Unlike the previous work, the authors use a two-step attention method which consists of a general attention mechanism and attention over return types, variables and methods from the environment. The authors tested their model against multiple baselines from previous works. With this, they found their model had improvements in both exact match accuracy and BLEU score.

2.4 Utilizing Transformers

In another piece of related work Kusupati and Ailavarapu (2018). converted natural language into code using a transformer model. This does not use recurrent structures like RNN and LSTM, and hence is different from the encoder-decoder models. The transformer uses multiple attention heads as well as a self attention component to compute the hidden vector representation of a sequential input. The authors experimental study on empirical data shows that the transformer approach can outperform a basic encoder-decoder approach; however, they do not compare to some of the more recent work discussed above.

3 Method

3.1 Data Source

All of our data is mined from the Stack Exchange Data Dump[2]. This is accomplished by identifying which snippets of code within an answer are most likely to fulfil the intent described by the title of the question. This task is done using reinforcement learning and requires a small gold standard set to train a model before it can be used to automatically mine more data. In particular, Yin et al. (2018) used a gold standard dataset with 330 entries for Java and 527 entries for Python. This gold standard dataset must be in the language for which we are interested in collecting additional data. We do not have a gold standard dataset for Spanish intents. The resulting data collected from Stack Overflow consists of the title of a question from Stack Overflow (the intent) and a snippet of code taken from one of the answers to the question.

3.1.1 CoNaLa

We are using the CoNaLa dataset as our baseline, as it is most similar to the Spanish data we collected. The dataset is divided into two distinct parts. There is manually curated data that are generally high quality. In addition to the original title of the question, this curated data includes a rewritten title that makes the exact intent of the question clearer and should make learning easier.

The second part of the dataset contains data that has been automatically mined from Stack Overflow using the process described in Yin et al. (2018). This portion of the dataset is considerably larger than the manually annotated data, which is obviously advantageous. However, it has not been reviewed, so it is possible that it contains erroneous entries.

In this paper, we use both parts of the CoNaLa dataset. The curated CoNaLa dataset consist of 2,379 training examples and 500 test examples. The automatically mined dataset consists of 16,880 training examples and 4,453 test examples. Two samples entries from the curated CoNaLa dataset are shown in Figure 1a and the full dataset is available online[3].

3.1.2 Spanish Translated CoNaLa

We created another dataset by translating the intents and the rewritten intents from the CoNaLa dataset to Spanish using the Google Translate API[4]. However, there are limitations to this dataset. The variable names remain in English, since we could not translate the code to Spanish. There could also be idiosyncrasies from the translation system introduced into the data that would not be present in a native speakers use of the language. These idiosyncrasies could be learned by the model we use which would decrease its usefullness when applied to queries from native speakers. A key benefit of this dataset is that it is much larger than the dataset we were able to mine from the Spanish language Stack Overflow. We used the same 2,379 training examples and 500 test examples as discussed above.

3.1.3 Mined Spanish Data

The mined Spanish data consists of 94 training examples. Because of the low number of training examples, we used the same 94 examples as the test set. Two samples from this dataset are shown in Figure 1b and the full dataset is available on GitHub[5]. Note that since this dataset was collected automatically from the Stack Exchange data dump, it does not have the `rewritten_intent` field that is included in the curated CoNaLa dataset. The consequence of this is that the correspondence between the intent and the code snippet may not be as clear and therefore be harder for a model to learn.

We initially attempted to collect the dataset using

[2]https://archive.org/details/stackexchange

[3]https://conala-corpus.github.io/

[4]https://py-googletrans.readthedocs.io

[5]While this paper focuses on the Spanish dataset, the repository contains data collected using the same technique for Portuguese, Russian, and Japanese in addition to Spanish. https://github.com/jackastner/so-nlcode-multilang

```
{
 "intent": "How can I send a signal from a python program?",
 "rewritten_intent": "send a signal `signal.SIGUSR1`"
  + "to the current process",
 "snippet": "os.kill(os.getpid(), signal.SIGUSR1)",
 "question_id": 15080500
},{
 "intent": "Decode Hex String in Python 3",
 "rewritten_intent": "decode a hex string '4a4b4c' to UTF-8.",
 "snippet": "bytes.fromhex('4a4b4c').decode('utf-8')",
 "question_id": 3283984
}
```

(a) Curated CoNaLa Dataset

```
{
 "intent": "Como llamar a una variable desde template django?",
 "snippet": "{{request.user.profile.lab.nombre}}",
 "question_id": 240978
},{
 "intent": "¿Cómo puedo hacer lo mismo que random.randint"
  + "pero con números reales?",
 "snippet": "random.random() * 20 - 10",
 "question_id": 241805
}
```

(b) Mined Spanish Dataset

Figure 1: JSON formatted samples from the CoNaLa and mined Spanish datasets

the techniques presented by Yin et al. (2018); Yao et al. (2018). These papers both use neural network based techniques to predict the likelihood that a snippet of code corresponds to the answer to some question. These classifiers are applied to Stack Overflow data to determine what snippet of code extracted from an accepted answer is most likely to answer the question posed by title of the original post. Yin et al. (2018) was used to to construct the original CoNaLa dataset. Unfortunately, we were not able to able to successfully apply either of these techniques to the Spanish Stack Overflow dataset.

Instead, we use an adaptation of an older data collection technique (Iyer et al., 2016) that relies on a series of heuristics. In particular, it finds accepted answers that contain exactly one snippet of code where this snippet is one line long after removing boilerplate code such as module imports and function headers. The original implementation of this technique also used a support vector machine (SVM) classifier to filter for "how to" question: questions that ask how to do something rather than asking for an overview of a topic or some other type of question that is less likely to be answered by a single snippet of code. Using this filter removed too many candidates from our dataset. Since our dataset was already considerably smaller than we would like, we removed the filter to have a larger if slightly lower quality dataset.

3.2 Preprocessing Data

Before the data can be input to the encoder/decoder model, the data must be preprocessed and loaded into custom *Dataset* objects. We modified a script from Yin and Neubig (2018) in order to do this. We pre-process the data the same way as described in Yin and Neubig (2017, 2018). This includes tokenizing the natural language intent snippets using NLTK (Bird et al., 2009), and replacing uncommon words with a `<unk>` token. The custom *Dataset*

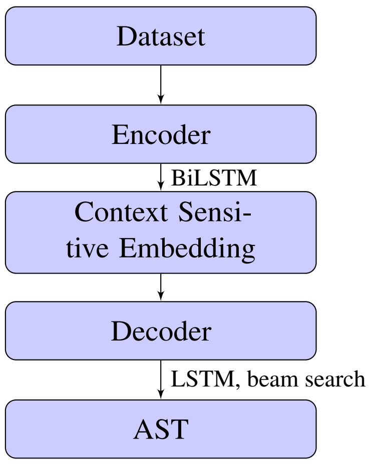

Figure 2: Overview of NL2code and TranX pipeline.

objects consists of two parts: vocab and examples.

The vocab is a dictionary of all the words found in the natural language snippets, along with their frequency counts. For the CoNaLa dataset, the vocab consists of 661 unique words. After translating the CoNaLa dataset to Spanish, its vocab consists of 757 unique words. Our mined Spanish dataset consists of only 54 unique words.

Each example consists of a tokenized version of the intent, the AST representation of the code, the raw text of the code, and the reference action sequence produced from the AST according to the grammar model.

3.3 Encoder Decoder Model

Initially we used the model from original NL2code (Yin and Neubig, 2017) paper which is based on an encoder/decoder model. For encoding, it uses a bi-directional long short-term memory (BiLSTM) network. The decoder is a conditional LSTM with attention. Along with the decoder, the code uses a beam search with a size of 15 to pick the best rule from the grammar as output for the AST. To create the model, the code uses the neural network library `nn` which is built on top of TensorFlow. The flow of data through this model is shown in Figure 2.

We also used the successor of NL2code, TranX (Yin and Neubig, 2018). The TranX model comes from the same creators of NL2code. Because of this, both models take as input very similar data objects, so our preprocessing step did not require any major changes. The encoder/decode part of the model is almost the same from NL2code (Yin and Neubig, 2017) except with the addition of parent feeding in the decoding stage. The parent feeding vector contains information about the parent's embedding and the state of its constructor from the reference action sequence.

4 Results & Analysis

We evaluate the generated code against the reference code using Bilinear Evaluation Understudy (BLEU). These are the same metrics used in the original NL2code (Yin and Neubig, 2017) and TranX (Yin and Neubig, 2018) papers. We use token-level BLEU since it produces a higher score when more sub-components of the generated code match the ground truth code. The results of our evaluation are summarized in Table 1.

4.1 CoNaLa - NL2Code

We first ran the NL2Code model on its previous datasets and got results matching to the original paper. We had trouble getting the same level of results with the curated CoNaLa dataset. The BLEU score is 17.8 with the CoNaLa dataset, which is much lower than the BLEU scores from the original paper (Yin and Neubig, 2017). In this paper, the HearthStone dataset produced a BLEU score of 75.8 and the Django dataset produced a BLEU score of 84.5.

One problem was that NL2Code was designed to generate Python 2 code, while CoNaLa dataset is designed for Python 3. This caused problems because the code was not able to create accurate ASTs for some examples. Another issue was that the grammar in the CoNaLa dataset is very different from the previous datasets used with this model. The HearthStone, Django and IFTTT datasets are all domain specific, while the CoNaLa dataset contains general purpose Python code snippets. We believe this made it harder for the model to learn from the examples. Because of these issues, we decided to use the TranX model for all subsequent runs.

4.1.1 CoNaLa - TranX

The CoNaLa dataset is one of the original datasets that runs with the TranX model. Because of this, we did not need to make any changes to train with the CoNaLa dataset and we were able to achieve similar results to the paper. Our result was a BLEU score of 24.05; the result from the authors Yin and Neubig (2018) is 24.5.

One possible reason that the BLEU score is better than NL2Code on CoNala dataset is how it pre-process variables and values of the intent. In CoNala, variables and values are distinguished by quotes. TranX has a slot-map where it stores those distinguished variable and values then later replace them in AST. So even if the predicted code had different functions than the referenced code, it was able to match the values and variables.

4.2 Spanish Translated CoNaLa

The Spanish CoNaLa dataset resulted in a BLEU score of 15.4 which was significantly lower than the English CoNaLa dataset. This could indicate that TranX is not able to perform as well on Spanish data as on English data; however, this Spanish data is the output of machine translation, so it may be not representative of real Spanish data.

4.3 Mined Spanish Data

The mined Spanish data preformed the worst. This is most likely due to the considerably smaller size of this dataset when compared to the CoNaLa dataset. The 94 pairs of natural language intents and code snippets are very unlikely to be sufficient for training any model. Additionally, the quality of this dataset may be lower than the CoNaLa dataset because we gathered the data without any gold standard Spanish language/code pairs.

The BLEU score after training and testing TranX on the mined data was 0. We suspected this was largely due to a lack training data. To confirm this, we tested the TranX model trained using the translated CoNaLa dataset on the mined Spanish data. This resulted in a slightly higher BLEU score of 2.58. While this is better than the 0 BLEU score obtained using only the mined data, it is still noticeably lower than the score when testing the model on the translated CoNaLa dataset.

One possible reason for this anomaly is that translating the English CoNaLa to Spanish introduced artifacts that are learned by the model, thus inhibiting performance on real Spanish data.

Dataset	Model	BLEU Score
CoNaLa	NL2Code	17.8
CoNaLa	TranX	24.05
CoNaLa — Spanish Translation	TranX	15.4
English Stack Overflow	TranX	2.63
Spanish Stack Overflow	TranX	0
Spanish Stack Overflow	TranX — Trained on Spanish CoNaLa	2.58

Table 1: Token level BLEU score for each dataset.

It is also possible that our technique for mining Spanish data does not produce well aligned natural language/code pairs. That is, some of the pairs we mined may not correspond to an intent and code that accomplishes that intent. If this is the case, then we would need to improve our data mining technique, possibly by applying the newer techniques used by Yin et al. (2018); Yao et al. (2018). As mentioned earlier, we attempted to do this but were not successful.

To determine if our data mining strategy was at fault, we applied it to the English Stack Overflow data dump to obtain a new English dataset. We then trained and tested TranX on this dataset. The model achieved a similar BLEU score of 2.63. This shows that TranX was not able to learn from either dataset gathered using our mining technique, even though it was able to learn from the CoNaLa dataset, suggesting that the data mining technique is a problem.

4.4 Measuring Similarity with Moss

Since we felt that BLEU score might not be the best measure of code similarity, we also tried to evaluate our result by using Moss, a code similarity detection system proposed by Schleimer et al. (2003). For each model we converted every intend output to single line and aggregated all the outputs into a single file before doing the same for the models predicted output. We then used Moss to check for similarity between the file containing ground truth outputs and the file containing model outputs.

Even though Moss does not require syntactically correct code as input, it failed to detect any similarity between model output and the reference snippets. One possible explanation is that as Moss tries to tokenize each file and find identical pairs in some clusters, but each line of the code in our output is independent of the other, causing Moss to have difficulties detecting similarities.

5 Conclusion

This paper aims to analyse the performance of the TranX model for natural language to code translation on Spanish language queries through the use of two Spanish datasets: one obtained by applying machine translation to an English dataset and another obtained by mining data from the Spanish Stack Overflow website.

In our experiments we first found that TranX performs as well as expected based on the results from its paper when applied to the English CoNaLa data set (BLEU score 24.05). We then evaluated TranX on three different Spanish dataset: translated CoNaLa, mined Spanish Stack Overflow, and a hybrid dataset where the training set is drawn from translated CoNaLa while the test set is our mined Spanish dataset. The BLEU score obtained from these evaluations were 15.64, 0, and 2.58 respectively.

These scores are all lower than the score obtained on the original CoNaLa dataset; however, we are not reasonably able to conclude that TranX does not perform as well on Spanish data as it does on English data. We have identified potential issues with our data that could be the cause of the lower scores rather than the difference in language.

The Spanish translation of the CoNaLa dataset is the result of machine translation and, as such, is likely not representative of the language a native Spanish speaker would use. The dataset mined from the Spanish Stack Overflow should be much more representative of how Spanish is actually used, but this dataset is too small to be useful in training the model. We also have concerns about the accuracy of the data collected by our method. The hybrid approach is perhaps the most reliable as it is evaluated on real examples of Spanish queries while still being trained on a sufficiently large dataset. On the other hand, this approach inherits the problems associated with both of our datasets.

It has the opportunity to learn idiosyncrasies of the machine translation process, and it is evaluated on a relatively small and potentially inaccurate test set.

References

Steven Bird, Ewan Klein, and Edward Loper. 2009. *Natural Language Processing with Python*. O'Reilly Media, Inc.

S. R. K. Branavan, Harr Chen, Luke S. Zettlemoyer, and Regina Barzilay. 2009. Reinforcement Learning for Mapping Instructions to Actions. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 1 - Volume 1*, ACL '09, pages 82–90, Stroudsburg, PA, USA.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. 2016. Summarizing Source Code using a Neural Attention Model. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2073–2083.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, and Luke Zettlemoyer. 2018. Mapping Language to Code in Programmatic Context. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1643–1652, Brussels, Belgium.

Uday Kusupati and Venkata R.T. Ailavarapu. 2018. Natural language to code using transformers.

Maxim Rabinovich, Mitchell Stern, and Dan Klein. 2017. Abstract Syntax Networks for Code Generation and Semantic Parsing. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1139–1149, Vancouver, Canada.

Saul Schleimer, Daniel S Wilkerson, and Alex Aiken. 2003. Winnowing: local algorithms for document fingerprinting. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 76–85. ACM.

Rolf Schwitter and Norbert E. Fuchs. 1996. Attempto - From Specifications in Controlled Natural Language towards Executable Specifications. In *In Proceedings: EMISA Workshop 'Natuerlichsprachlicher Entwurf von Informationssystemen - Grundlagen, Methoden, Werkzeuge, Anwendungen*.

Stack Exchange Inc. 2019. About Stack Overflow.

Yuk Wah Wong and Raymond J. Mooney. 2006. Learning for Semantic Parsing with Statistical Machine Translation. In *Proceedings of the Main Conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, HLT-NAACL '06, pages 439–446, Stroudsburg, PA, USA.

Ziyu Yao, Daniel S. Weld, Wei-Peng Chen, and Huan Sun. 2018. StaQC: A Systematically Mined Question-Code Dataset from Stack Overflow. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web - WWW '18*, pages 1693–1703, Lyon, France.

Pengcheng Yin, Bowen Deng, Edgar Chen, Bogdan Vasilescu, and Graham Neubig. 2018. Learning to Mine Aligned Code and Natural Language Pairs from Stack Overflow. In *Proceedings of the 15th International Conference on Mining Software Repositories*, pages 476–486.

Pengcheng Yin and Graham Neubig. 2017. A Syntactic Neural Model for General-Purpose Code Generation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 440–450, Vancouver, Canada.

Pengcheng Yin and Graham Neubig. 2018. 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*, pages 7–12, Brussels, Belgium.