Speaking Robot: Translating Natural Language to RLang

Gabriel Rizk
gabriel_rizk@brown.edu

Brandon Li
brandon_li@brown.edu

Timothy Park
timothy_park@brown.edu

Abstract

RLang is a newly developed Reinforcement Learning declarative language that helps alleviate the difficulty in expressing prior knowledge in a way that is accessible to learning agents in the field. However, with RLang being a novel language in a developing field, development with Markov Decision Processes (MDPs) and reinforcement learning (RL) can remain inaccessible to developers without specialized knowledge and training. We propose a language model capable of translating natural language into valid RLang, as well as error metrics that are capable of evaluating this model's performance.

1 Introduction

For Artificial Intelligence and intelligent agents to successfully take on more advanced and complex tasks in the future, humans and AI must be able to communicate more naturally and efficiently. The current methodology to train intelligent agents through reinforcement learning (RL) is certainly limited. In terms of a Markov Decision Process (MDP), even a small amount of information about the world and previous states would drastically improve learning, especially in complex, real-world environments. A uniform RL language that both humans and AI could translate to and from is an essential next step in improving Reinforcement Learning for intelligent agents.

Currently, feeding natural language (NL) instructions to an intelligent agent and having the agent process, understand, and retain the full context and meaning of the instructions is challenging. Natural language instructions can take on many complex forms and do not always adhere to concrete sentence formats that an intelligent agent could easily parse through. Because of the wide range of possibilities of sentences and meanings that could be expressed with NL, grounding NL in MDPs using a Reinforcement Learning language can lay the foundation for clearer communication between humans and an intelligent agent. Training an intelligent agent to ground NL in MDPs using a RL language will help the agent eventually break down and understand the non-sequential composition of NL better.

In this work we focus on RLang, the most notable RL languages in terms of expressive power to be used to ground NL in MDPs [6]. Under the criteria of gauging expressive power when expressing prior knowledge of a MDP: policy hint, action structure, policy constraints, state structure, rewards, values, and transition dynamics, RLang stands above its competitors such as ALisp [1] and AdviceRL [5]. If humans could communicate to AI through a uniform RL language like RLang, the degree of freedom and creativity for tasks that could be assigned to AI could be leaps more expressive than current standards. RLang will be the first and essential step in enabling natural language communication with intelligent agents and eventually deploying these agents in the real world.

2 Background

2.1 Machine Translation

There has been extensive research done on translating Natural Language to machine code, instructions, or even database queries in SQL [4]. In this work, we focus on translating Natural Language to a Reinforcement Learning language, RLang. We draw on techniques used in previous works on Natural Language Translation such as token parsing and take advantage of the bidirectional nature of machine translation [2]. More specifically, we first generate the RLang dataset, parse the RLang, and then translate it to NL, but we train our model to translate from NL to RLang.

2.2 MDPs Markov Decision Processes (MDPs)

MDPs are defined by a tuple (S, A, T, R, p, γ). S denotes the finite number of states in a subset of R^n. A denotes the set of actions that an agent could take. R denotes a task-specific reward function, p denotes the initial state distribution, and γ denotes the discount factor. In this work, NL will be grounded in the context of MDPs using RLang.

2.3 RLang

RLang is a declarative Reinforcement Learning language to naturally and unambiguously express prior knowledge about different parts of a MDP. RLang is a very sequential language with specific grammar structures that allow fluid expression of transition dynamics, rewards, values, state-features, and policies of a MDP. The main elements of RLang are separated into State Factor, State Feature, Proposition/Symbol, Policy, Sub-policy, Markov Function, and Effect Expression. RLang is inspired by Planning Domain Description Languages (PDDLs), which, similar to RLang, provides a complete domain of factored-state environments such as actions and its effects. While PDDLs describe transition functions for planning, RLang provides partial information to a reinforcement learning agent in any component of a MDP to better train the intelligent agent [2].

3 Methodology

To generate a large corpus of valid RLang statements that map to understandable natural language for our training and evaluation datasets, we used the Python-based natural language toolkit NLTK [3] script for RLang generation, tokenization, and translation into natural language.

3.1 Generating RLang Dataset with Context Free Grammar

We started by defining and writing Context Free Grammars (CFGs) for each of the MDP components: constant, effect, markov feature, predicate, subset, option, and policy. We initially thought it was best to utilize all MDP components for our RLang dataset to fully capture the richness and diversity of Natural Language after translation. Based on the CFGs, we produced up to 100,000 RLang statements composed of valid combinations of the different parts of the statements for each MDP.

3.2 Tokenizing RLang Sentences

With the generated valid RLang statements for each MDP component, we parsed the RLang statements into tokens based on its corresponding CFG for easier translation between RLang and NL. Because RLang is a very sequential language made up of distinct partitions, it was easy to parse through the RLang statements, separate each part into tokens, and store the tokenized form of each statement.

3.3 Translating Tokenized RLang to Natural Language

To transform the RLang tokenized statements to English NL statements, we defined a set of synonymous possible translations for each token to allow for more flexibility in interpretation. Based on these translations, we also reorganized the ordering of phrases when applicable to allow for more natural English grammar. Through this process, although we started off with all MDP components, we ultimately decided to only use the MDP components, option and policy, for our research. The other MDP components such as effect and markov feature proved too difficult to properly express in NL. Expressing state changes and defining features in MDPs with English required awkward wording and phrases that humans would never normally use.

3.4 Preprocessing

Standard preprocessing for natural language tasks did not apply here, so our preprocessing stage was slightly different. As we are working with a fairly small vocabulary size, we did not need to stem or lemmatize our vocabulary. Since our corpus of text was procedurally generated, it did not guarantee that our produced English was syntactically valid English. As a result, we had to preprocess some of the sentences to ensure that the English produced was valid.

3.5 Model Architecture

For our primary model, we used the original transformer architecture [7] to solve the machine translation task of translating natural language to Rlang. We used 8 attention heads, similar to the original paper, but used an embedding dimension of 256 to save computation time. Our performance was not hindered by this, due to the small size of our vocabulary corpus. Aside from this difference, our model follows the specification set out in “Attention is All You Need” [7]. The transformer architecture is shown below.

We at first built an LSTM to produce output, but realized that the results produced from a relatively small LSTM model would not be sufficient to produce compilable RLang. We have included the results of the LSTM below as a reference to demonstrate that producing RLang statements from natural language is a difficult task that requires state-of-the-art models to properly solve. LSTMs fail when sequence lengths are long due to vanishing and exploding gradients over the time sequence. At an average length of 23 words per RLang expression, the sequence appear to be too long for an LSTM to handle. Nevertheless, we have kept the results of our LSTM model below to demonstrate the difference in performance.

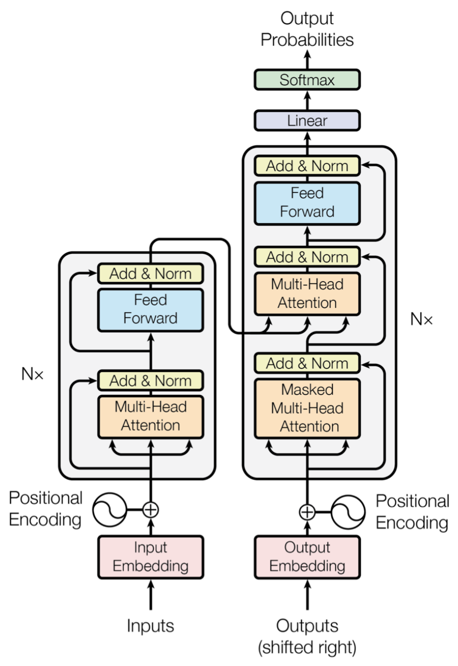

Figure 1: The Transformer - model architecture.

Figure 1: Transformer Model Architecture

4 Results

Because we were using a more limited dataset of self-generated RLang in which each token had a small subset of possible words of phrases, we propose two new metrics for evaluating the performance of our model's ability to translate natural language into RLang: Simple Token Similarity (STS) and Boolean Correctness (BC).

4.1 Simple Token Similarity

We propose STS, where RLang statements are compared via tokens rather than by individual words, as we wanted to penalize the logical mapping of the statement over the individual character edit distance between words. To do so, we parsed both the target RLang and the model-translated RLang into trees of tokens and calculated the Zhang-Shaha tree edit distance between both [**?**], where each token represents a node, and changing the label of such a node counts as one edit. In cases where the RLang was not a valid and compilable statement, the entire reference sentence (which generally averaged about 23 words) would be penalized per word.

4.2 Boolean Correctness

Boolean Correctness (BC) is an extension on STS as we realized that one of the most common errors we encountered in the model translation was in its boolean expressions where, for example, policy statement would check if two variables would be equal or $==$ when the natural language actually had meant to check if one quantity was greater than the other or $>$. We penalized this statement

Expression Type	STS	BC
Policy	0.31	1.52
Option	0.27	1.59
Both	0.29	1.55

Table 1: Transformer Results

Expression Type	STS	BC
Policy	22.7	23.4
Option	22.9	23.3
Both	22.8	23.3

Table 2: LSTM Results

by counting each token encapsulated within check statement as ‘wrong,’ as logically the statement would no longer encapsulate the same subset of cases as the original intended subset (For example, instead of capturing all elements of a set that are greater than a, the statement would instead capture all elements of a set that are equal to a).

4.3 Experiments

To produce our evaluation dataset, we randomly selected a test dataset of 20,000 example sentences from the training dataset to be taken out of training and be used for testing. The results of the model are shown by the table in figure below. Our LSTM failed to produce valid RLang over 97% of the time. Of the two models evaluated, the transformer with attention model performed far better than the LSTM model, translating 99.4% of the example sentences into compilable RLang, and translating 47.8% of the sentences with perfect accuracy. All results shown below are in relation to the Transformer model, as the LSTM model failed to produce reasonable output that could be properly evaluated.

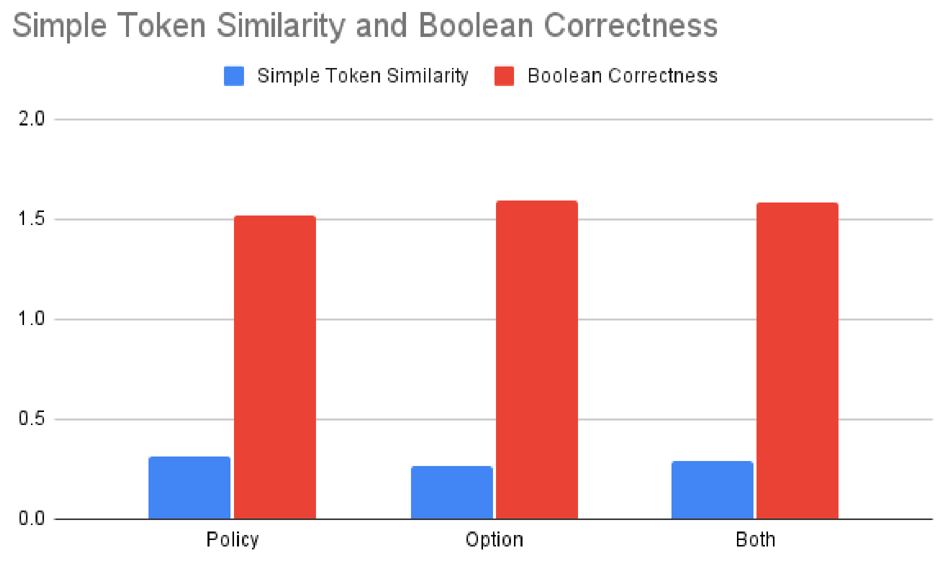

Figure 2: STS and BC for Transformer

4.4 Qualitative Analysis

Issue	Natural Language (input)	Expected Rlang	Generated Rlang
Flipped Boolean	a entails that you do find if b is greater than or equal to y , or do run as a last resort	`Policy a :=` `if b >= y :` `Execute find` `else :` `Execute run`	`Policy a :=` `if b <= y :` `Execute find` `else :` `Execute run`
Invalid Rlang	a is where if x is greater than or equal to 0 you have the choice to run until finally b is equal to 2	`Option a :` `Init x >= 0` `Execute run` `until b == 2`	`Option a :` `Init x >= 0` `Execute run` `until b run` `Execute run` `Else != 2`

Figure 3: Transformer Issues

The above breakdown provides us with an estimate of where our model is producing mistakes on the generated RLang statements. It seems that our error lies in the incorrect generation of boolean operators ($==$, $>$, $\leq$, etc.). The increase in scoring for the Boolean Token Difference is due to categorizing the statements below the boolean operator as incorrect. Our justification for this scoring system is that, once the boolean expression above it is incorrect, all words below the if statement in the tree should also be categorized as incorrect as the RLang does not execute properly. Let us look at some examples of generated text to provide a qualitative analysis. The table above shows two issues that seemed most common in the data generated by the Transformer.

The first issue shown (and the most common issue) is that the boolean expression was flipped from greater than or equal to to less than or equal to. From investigating the results of other flipped boolean expressions, it seems that the positioning of the variable in the natural language may have caused this boolean to flip. Longer training or more attention heads could cause this issue to occur less frequently.

The second issue shown, which is much less common than the first issue, is that invalid Rlang was produced. Investigations into why this occurred seem to stem from input natural language statements, as the model may have gotten confused with the word “finally”, expecting another statement. Other data points that have produced similar errors have also had “finally” produce an extra expression. This is likely due to sparsity in our vocabulary, as “finally” seems to be closely related to creating a new expression, rather than be used as a terminating expression for the statement.

5 Discussion and Future Work

Our project intended to find a way to ground simple human readable natural language to RLang, such that RLang can be used to express instructions to an agent. We have done this in two steps. Firstly, we have created a pipeline to generate CFGs that map natural language to RLang statements. Secondly, we have created a Transformer model that takes in natural language, and translates that into an RLang statement. We have shown that our model does indeed work fairly well, as we are able to go from our corpus of natural language to a compilable and correct RLang expression. As a result, we have shown that it is possible to ground simple human readable natural language to RLang. Although our model and pipeline can still be improved upon.

There are a number of ways that we can improve on this project. The primary source of improvement will likely come from improving the variability in our data. A larger corpus of data with a more varied vocabulary would allow us to generalize the produced RLang better. Natural language that is procedurally generated will very frequently have odd phrasings that seem unnatural, as is shown in our data. A further step would be to optimize our context free grammars to produce data that sounds more natural. Other advancements should include adding more RLang expression types to this model. We created CFGs and produced data for other RLang expressions such as Effect and Markov Feature, but decided against using them in our model due to the awkward phrasing that our CFGs produced for expressions related to state changes. In the future, we would like to add more RLang expressions to our model to determine if we can build a full-scale translator.

References

[1] D. Andre and S. J. Russell. State abstraction for programmable reinforcement learning agents. In *Eighteenth National Conference on Artificial Intelligence*, page 119–125, USA, 2002. American Association for Artificial Intelligence. ISBN 0262511290.

[2] A. C. Anush Kumar1, Nihal V. Nayak2 and M. K. Nair1. Study on Unsupervised Statistical Machine Translation for Backtranslation. 2009.

[3] S. Bird, E. Klein, and E. Loper. *Natural Language Processing with Python*. O'Reilly Media, 2009.

[4] R. Cai, B. Xu, X. Yang, Z. Zhang, Z. Li, and Z. Liang. An encoder-decoder framework translating natural language to database queries. 2017.

[5] R. Maclin and J. W. Shavlik. Creating advice-taking reinforcement learners. *Machine Learning*, 22(1):251–281, 1996.

[6] B. Spiegel. Rlang: A declarative language for expressing prior knowledge about mdps.

[7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. 2017.