Automatic Translation of Natural Language to SystemVerilog Assertions

Abhishek Chauhan

Agnisys Technology Pvt. Ltd., abhishek.ch@agnisys.com

Abstract

In a design development cycle, a large amount of time and resources are consumed in the very tedious cycle of verification/validation of the design against the design specification, leaving behind a shorter time-to-market. There is a huge demand and growth for automation in this aspect of the development cycle. Often verification engineers are required to formally validate a design against its actual specification through assertions that are fed to a formal tool. These assertions are tricky to write, especially if the engineers have less experience with writing assertions. Machine Learning (ML) algorithms can be used to eliminate this void and automate the manual work of capturing accurate assertions against a design specification and thus save time as well as other resources.

We propose a novel supervised approach for a machine translation of natural language to SystemVerilog Assertions (SVA), which is trained using self-generated datasets. Various Natural Language Processing (NLP) augmentation techniques are used in this approach to increase the dataset such as back translation, random noise addition, synonyms replacement, etc. which helps in model generalization. The objective is to train the model with smaller subsets and then increase the model generalization with further augmentation techniques.

Introduction

SystemVerilog Assertions (SVA) are primarily used to validate the behavior of a design against its original specification. SVA are sometimes used to provide information on functional coverage for a design and may also be used as a formal specification language, making the requirements clearer and more unambiguous, and also making it possible to automate validation of the design.

Writing SystemVerilog Assertions that are compact yet accurate is a difficult task, and often the actual written SVA may not be equivalent to the intended assertion/check, leading to missed design bugs or lower functional coverage.

This paper aims at solving this problem of writing such sophisticated SystemVerilog Assertions with the help of machine learning (ML) by bridging the gap for the verification engineers who do not have prior knowledge of writing SVA to functionally validate a design. The idea is to create a system to understand the user's natural language description of assertions and generate SVA, and vice-versa. This will help users to conceptualize their intended assertions as actual SVA and to understand the SVA in the code.

Machine Learning space used

Related work

With a one-step attention Recurrent Neural Network (RNN) based network, the model performed poorly due to issues like name generation, Out-Of-Vocabulary (presence of ample names) and considerably long sentences. The CRF (Conditional Random Field) based RNN model, which was also tried for segmentation of sentences and Named Entity Recognition, performed quite well. The idea of considering it was dropped later as it does not support the latest Tensorflow library and will need extra dependency on a third-party library.

We later tried implementing the copy mechanism in a one step attention network and the model was found not to be able to comprehend all names in the input sequence and suffered drastically due to word repetition.

We also explored VAEs (Variational AutoEncoders) to learn text representation in space and thus by using representation we tried to generate a syntactic dataset. This generated a poor dataset and so was not considered further. As to train VAEs, there is a requirement of larger dataset for representation learning.

Actual ML application

Transformer neural network

Transformer neural networks are the state-of-art networks in Natural Language processing. They have the capability to out-perform all NLP datasets as compared to earlier models such as SQUAD dataset, BABL, Language translation parallel corpus, Stanford Sentiment Treebank, etc. They are also used in Computer Vision as they perform way better than the earlier networks. This type of network shows a promising future in neural nets task achievement. Development has been going on to optimize model training and inferences on hardware as the size of such networks is constantly growing. The network involves usage of simple MLP with fully connected layers and attention operation, which makes it very simple.

In the machine translation task, to translate English natural language into SystemVerilog Assertions, there was an issue with name generation where we used the ‘Named Entity Recognition (NER)’ model to deal with the problem. The NER basically extracts entities based on learned context representation, which involves usage of only the encoder and the final layer as a fully connected layer to predict each token in the input sequence. On the other hand, the encoder with added decoder was used for inter-language translation. The network was trained using supervised techniques.

A parser-based machine translation technique is used, which basically segments the natural language based on SystemVerilog delay clock cycles, and then uses those segments to form separate embedding to impute this useful information in the machine translation scenario.

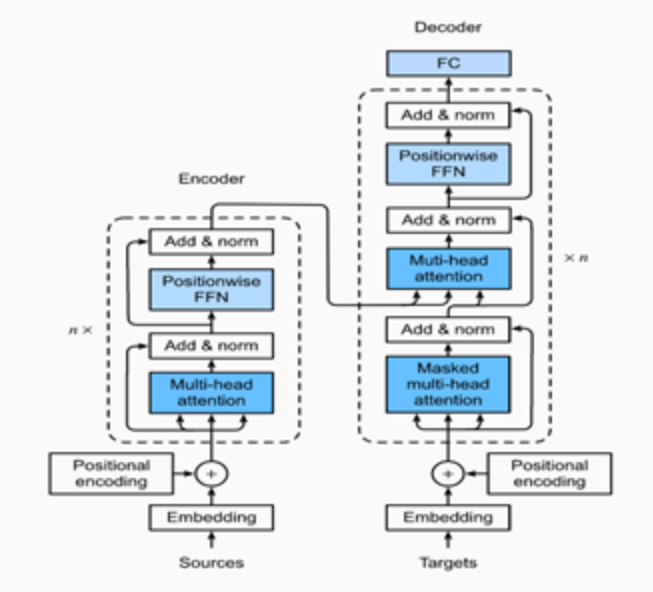

Figure 1: Transformer neural architecture (Ref. [9])

Tokenizer

To deal with Out-Of-Vocabulary words and with less availability of the dataset, we used a pre-trained vocabulary English (bert-uncased) Bert-Tokenizer and also fetched the pre-trained embeddings for the same. This tokenizer works in a manner that breaks words into multiple sub-words to handle text including the unknown texts. It does not consider multiple spaces, and also separates punctuations with a space.

Data collection

For natural language, datasets are available but those are only suitable for learning the representation for English language in an unsupervised fashion and no dataset is available for SVA as of now. This task involved domain specific cases. We explored and researched the domain specific sites for SystemVerilog Assertions but found very little information in terms of public usage. Thus, as a result a huge amount of time was invested in data mining.

We considered generating datasets by referring to the rules of SystemVerilog assertions syntaxes (SystemVerilog 3.1a Language Reference Manual Ref.[1]).

We then created the collection for phrases dictionary for SVA syntaxes, such as predefined properties, clock cycle delays, Boolean operators, comparison expressions, repetition operators, implication, etc. Python scripts created on the top of SystemVerilog Assertions defined rules for writing those script rule-based assertions. They were also used to verify the combination of the syntaxes, operators, and their corresponding English phrases to generate target sequences. These data generators involved complex mapping of rules to adhere to the rules of the target language to produce the correct form of English. Recursive placeholder replacement was also used for nested SVA expressions. We added annotation checks, and error assertions to ensure correct sample generation. Creating the phrase collections was laborious and time consuming. We collected the initial data samples from multiple sources including industry standards to focus more on the practiced/used SVA.

Further data generation

The collection of samples needed more additions. We used the NLP augmentation technique which is Back Translation. The technique involves the use of a trained machine translation model that helps convert input sentences (English in our case) to many other selected languages like German, French, Spanish, etc. and then retranslate from the respective output language to the initial language (English). Basically, the techniques helped to generate new phrases with almost the same meaning as an input sequence. We observed that by using back translation with a particular language, new samples were obtained with proper restructuring of input sentences to a completely different sentence yet with the same meaning. But this process needed to be continuously monitored and analyzed as the sentences could sometimes distort and generate completely irrelevant sentences.

Data inconsistency

Data inconsistency was observed in large numbers when the dataset was created manually. Large number of samples had overlapped meanings, like the same phrase mapped to multiple outputs. Any model will find it difficult to

converge with such an inconsistency. The debugging of such a case is a laborious task and consumes a lot of time.

Data inconsistency was also found in general cases such as in case of machine translation from English to German and vice versa, which can be defined as parallel corpus as one word corresponds to combination of words in other languages and in some cases even word to word mapping exists. But in this case, the dataset was not a parallel dataset as the sentences define a property or its relationships and the context for any signal name used.

Training

A batch size of 64 was used and a sentence length of 512 was generated as input for the machine translation model. This was trained for 50 epochs where each epoch consisted of 300 batches. The model trained quite well with very good accuracy. We observed that the model could learn easily even when complex sentences were mapped. Currently, the model is trained with a data centric approach due to the absence of a generalized dataset. The model is expected to overfit on such a dataset as we have observed and trained it with augmented samples.

Training with data augmentation

To avoid overfitting and increase model generalization to perform better on new samples, a lot of techniques were researched and implemented. Some of them were provided by the Nlpaug Python library (an NLP augmentation library) and below are some which were manually implemented to cover our scenario:

- word swapping - a random pair word was swapped in a sentence
- word insertion - a random word or combination of word were inserted in between a pair of words
- word substitution - a word was replaced with some other words (both good and bad words)

- synonyms substitution - in order to increase the sample space, words were replaced with their synonyms, and a Fill-Mask pretrained model was used to produce such words
- antonyms substitution - to contradict the meaning of a sample, words with opposite meanings were replaced with one or more words (such samples were created with much less probability, only to increase model dependency on other words)
- back translation - it was done using a Google translation API that proved to be very helpful in generating new samples [Table 1]

Sentences were augmented at character level, so that the model can handle the user's spelling mistakes as well. And so random character insertion/substitution/deletion, typing errors, etc. were also used and these utilities were provided by the Nlpaug library.

The model performed exceptionally well with these techniques, but it was observed that we could only augment a few samples in a single batch, rather than all for the model to be able to learn with accuracy.

English	French	English
signala is high and go low in the next cycle	signala est élevé et diminue au cycle suivant	signala is high and decreases in the next cycle
signala changes to 1	le signala passe à 1	the signala goes to 1

Table 1:Back translation sample

Experimental results

Hyperparameter selection

Hyperparameter can be considered as the number of layers, attention head, and embedding size that decides the fate of models. These parameters also affect the model training time, convergence time, and inference time. We considered 4 layers with 8 attention and 128 embedding sizes for the task after trying out a few other combinations. This scenario proved to be the best among our considerations.

S.no.	Hyperparameters	Augmentation	Remarks
1	{num_of_layers:4, attention_heads:8, embedding_size:128}	No	The model trains very fast and is able to converge within 25 epochs to 99.0 and achieved accuracy of 99.9 on further training. The model overfits.
2	{num_of_layers:4, attention_heads:8, embedding_size:128}	yes	The model is able to converge to 99.2 but keep fluctuation between the 98 to 99.2. This is because of adding noise using augmentation. This model generalizes well with new samples.
3	{num_of_layers:4, attention_heads:8, embedding_size:512}	No	The model becomes slow. And it is not able to cross 97.8. Vanishing gradient might be the problem. It took a long time to achieve this accuracy.
4	{num_of_layers:6, attention_heads:8, embedding_size:128}	No	Due to an increase in layers, model accuracy starts jumping with a large gap. It increases for a time then decreases then starts fluctuation. Data inconsistency can cause such problems. The model trains slowly.
5	{num_of_layers:6, attention_heads:8, embedding_size:512}	No	The model trains very slowly. Model accuracy increases gradually then decreases and starts jumping between 60 to 70. Vanishing gradients might be the issue. Also, data inconsistency can cause such problems.

Table 2:Hyperparameter selection

Factors such as data sample imbalance, overlapped meanings, and considerably long dependencies can also affect the accuracy of neural nets. Batch size on the other hand did not affect the accuracy as much and thus we considered only ‘64’ as the batch size. Due to the smaller labelled datasets and dependency on augmentation technique for model training, large models were performing as per expectation.

Input	Predict
signala is stable, signaln is low after signalg is unknown, after 2 cycles, signalb is low	$ stable (signala) $ isunknown (signalg) ! signaln # # 2 ! signalb
ack is stable , req is low after pready is unknown, after 2 cycles, aready is low	$ stable (ack) $ isunknown (pready) ! req # # 2 ! aready
After a 5 valid signala, signala goes high until signall arrives	$ rose (signala) | - > # # 5 (signala throughout (signall [- > 1]))
When signala deassert and after 2 clock ticks throughout which signale is low and it is for 10 cycles	! signala # # 2 throughout (! signale [* 10])
Whenever signala goes high, signalb must go high within 2 clocks and go low in the next clock	$rose (signala) |-> ## [1 : 2] signalb ##1 ! signalb
The signala is not equal to signald for 10 or more non consecutive clock cycle	(signala! = signald) [- > 10 : $]
signala rose to 1	$ rose (signala)
signala is high and go low in the next cycle	signala # # 1 ! signala
The sequence in which signala is high and signald is low and the sequence in which signalc is stable and in the next cycle, signale is low.	(signala & &! signald) and ($ stable (signalc) # # 1! signale)
The sequence in which signala is high and signald is low and the sequence in which signalc is stable and in the next cycle, signale is low. They must end at same clock cycle	(signala & &! signald) intersect ($ stable (signalc) # # 1! signale)
When signala is high and in the same cycle signalc is less than the value of signald before 2 clock ticks and must occur within the sequence signala is high.	(signala | - > signalc < $ past (signald, 2)) within (signala)

Table 3:Data samples with predictions

Results

The transformer neural network has come up as the state-of-art for machine translation with great accuracy with complex descriptions of assertions in natural language.

Both the translations, natural language description to SVA as well as SVA to standard English description, show accuracy of around 99% with the training dataset and around 83-85% on an internal test dataset.

Application

Verification/validation engineers can now easily capture SVA though their English assertion descriptions, and vice-versa, and can hook these automatically generated assertions to their environment for functional validation of the design.

A current trial version has been deployed at ‘www.ispec.ai’ for users to try and play with the model. Users can also provide feedback if the generated SVA deviates from their intended assertion description to further improve the efficiency of the model and make it more robust.

Future work

- Aiming to collect more data from industry sources to make the model more reliable and further improve its accuracy
- Beam search for text generation
- Train a larger transformer network with various objective tasks for complex natural language representation

References

[1] SystemVerilog 3.1a Language Reference Manual - extensions to the IEEE 1364-2001 - IEEE Standard Verilog Hardware Description Language

https://ieeexplore.ieee.org/document/954909?arnumber=954909

[2] Attention Is All You Need
https://arxiv.org/abs/1706.03762

[3] Transformer: A Novel Neural Network Architecture for Language Understanding
https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html

[4] Nlpaug python library

https://github.com/makcedward/nlpaug

[5] Incorporating Copying Mechanism in Sequence-to-Sequence Learning

https://arxiv.org/abs/1603.06393

[6] A Survey of Data Augmentation Approaches for NLP

https://arxiv.org/abs/2105.03075

[7] Automating the Translation of Assertions Using Natural Language Processing Techniques

[8] Generating Formal Verification Properties from Natural Language Hardware Specifications

[9] https://d2l.ai/chapter_attention-mechanisms/transformer.html