AI Based Sequence Detection

Asif Ahmad , Abhishek Chauhan
asif@agnisys.com , abhishek.ch@agnisys.com

INTRODUCTION

In this era of automation, this article aims to help in capturing verification and validation sequences by automatically generating them from Natural Language using Artificial Intelligence (AI) based sequence detection techniques and then using those sequences in C/UVM code. The article talks about the current state of development and gives ideas about how the reader can implement their own solution to achieve true specification driven software development.

OVERVIEW

With continuing advancement of Artificial Intelligence (AI) and Machine Learning (ML), their usage in the field of technology has increased by many folds. Their application has increased in diverse fields such as face detection technique, face wrapping, object detection technique, goal classifiers, language translation, chatbots, spam detection, data scrapping, etc. Through AI, rule-based applications have vanished and taken a back seat as there have been many algorithms invented which are capable of defining its own rules or create classifiers like linear Regression, Logistic Regression, Trees, SVM etc. Along with the Algorithms what is really important, is the data that is used to train the model of these algorithms. In EDA, the application of ML or Deep Learning techniques enables the modeling and the simulation with an unprecedent levels of insight, hence one can expect greater efficiency and accuracy from design tools which would definitely translate into shorter turnaround times and providing greater flexibility in analysis, simulation coverage and thus shift a step ahead towards broader automation. It can help in identifying patterns to optimize design, allowing the designers and the testers to model more complex design in simpler way and in lesser time and hence make the designs more efficient in multiple aspects of automation and generation of design as well as in verification and validation with assertions, sequences for special registers providing maximum testing report which can surely help in accelerating the entire design process in general. Also ML-generated models can provide better feedback to the designers or engineers by indicating whether the design would live up to the expected performance at each step of the development process.

Usage of Natural Language Processing (NLP)

The usage of Natural Language Processing (NLP) for manipulation of natural language text to determine and capture sequences is now possible using deep learning techniques. Applications like Machine Translation, Speech recognition, Conversational Chatbots, POS (parts of speech) Tagger have been most popular among the NLP applications.

NLP is basically a vast field of research area broadly used for determining speech or text written in communication language and this field grew out of the field of linguistics and has succeeded above expectation so far and even more is there to achieve in the near future . It can be used for the development of deep learning models for classification of text, translation of text and more. NLP is sometimes also referred as 'linguistic science' in order to include both classical linguistics as well as modern statistical methods. In ML, we are concerned more with the tools and methods from the field of NLP which is basically automatic processing of human understandable language.

Deep Learning is a subfield of machine learning that focuses on the algorithms inspired by the structure and function of the brain. These techniques have proved useful in solving challenging natural language processing problems. Several neural network architectures have great impact in addressing natural language processing tasks.

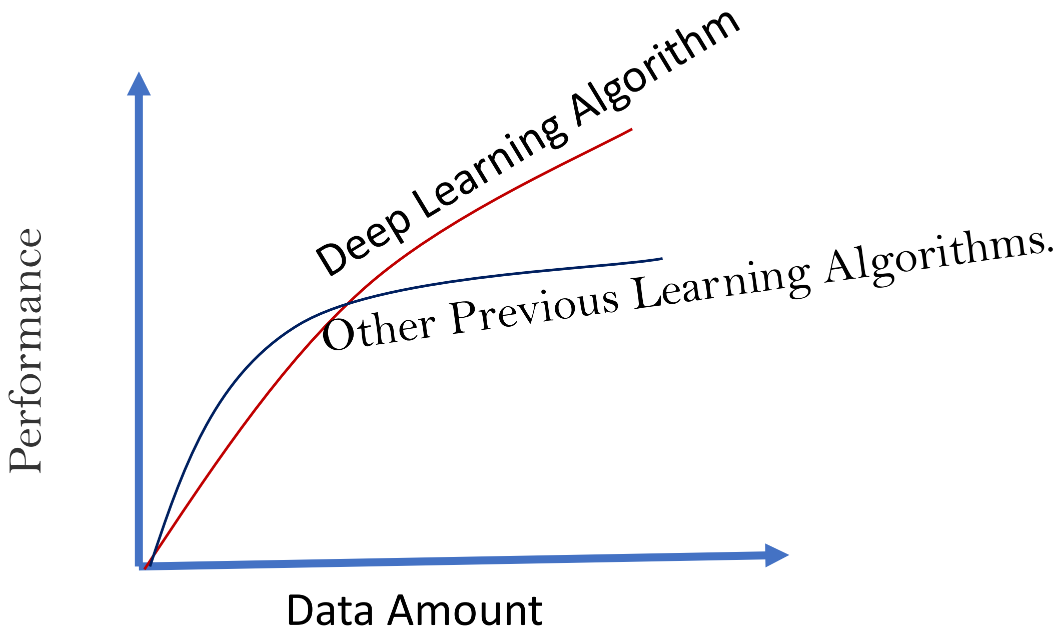


Figure 1 – comparison graph

Usage of Recurrent Neural Network (RNN)

A neural network is nothing but a series of algorithm that attempts to recognize basic relationships in a set of data similar to the way the human brain operates. Recurrent Neural Network (RNN) is one such type of neural network designed to deal with real world problems like machine translation, chatbot, etc. The input and output are connected as previous step output is fed as current step input to predict the new text. The RNN comes with a hidden state that basically remembers some information about a sequence with the help of a memory in it. It uses same parameters recursively for all inputs and performs the same task on them as well as the hidden layers to produce the output reducing the complexity of those parameters.

RNN is used in NLP to carry pertinent information from one input item in series, i.e., it can take a series on input without any predefined limit on size. Basically, in RNN every word gets transformed into machine readable vectors. This sequence of vectors are processed one by one. The processing is done by passing the hidden state to the very next step of the sequence and this hidden state acts as a memory of the neural network which holds the previous data seen by the network itself, i.e., all the inputs are related to each other. This makes it applicable to accomplish tasks such as unsegmented, connected handwriting recognition or speech recognition.

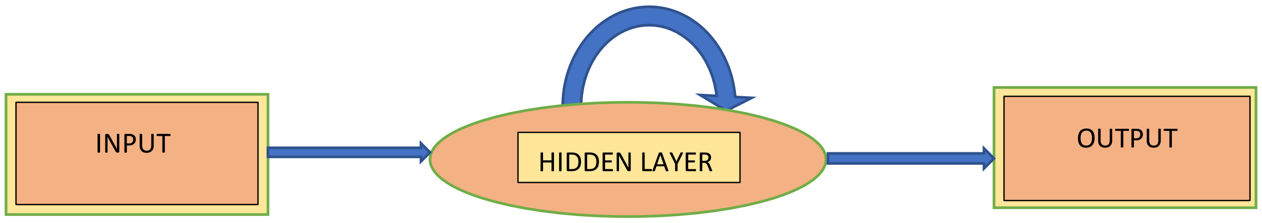


Figure 2- A typical RNN unit with a recurrent hidden layer

RNN Bases Long Short Term Memory(LSTM)usage

As RNN cannot process very long or large sequences, Long Short Term Memory (LSTM) is used which is a modified version of the RNN that makes it simpler to remember past data in the memory helping in retaining memory of long sequences.

This type of RNN is very well suited to classify and process as well as predict time series given the lags of unknown duration. It trains the model using back propagation and at every time step, attention is given to those words that direct towards the prediction of most part of the output. This attention weight is calculated using an algorithm and formula for each time step and then these attention weights are multiplied by each hidden state of respective time step to form attention values which is achieved using the attention mechanism that allows to focus on certain part of the input sequence while predicting a certain part of the output sequence that ultimately enables easier learning and higher quality of prediction. A final context vector is build by the dot product of all attention values which is also known as stacking. This context vector enables the decoder to focus on those certain parts of the input sequence while predicting its output.

Example: -

Let the context vectors be c1, c2, c3, ...

with h1, h2, h3, … be output vectors of the encoder

and α1, α2, α3, ... be their attention weights

So, the dot product would be -

$$c_i = \sum_{j=1}^{4} \alpha_{ij} h_j$$

The output is then fed word by word to the decoder of the LSTM network and at each time step context vector is used to produce appropriate output. Thus, the attention mechanism located between the encoder and the decoder enables improved performance.

Sequence Detection

Over the years we have come across and understood a lot of different ways people use registers associated with the Hardware/Software Interface (HSI). This information has helped us understand the kind of sequences users create to program and test their IPs.

As in an ideal world, users would rather use plain and simple English text to describe the sequences rather than encode in various languages. In any case, this is being done in the original specification. Natural, plain English is still the hallmark of specifications in today's system design and a lot of useful and actionable information is embedded in the natural language specification text.

Numerous translations happen when the architect/designer creates a specification in English language and the hardware/software/firmware engineer must manually convert them into code. With this new methodology, the specification writer's original intent is converted into real, usable code. They need to just describe the sequences in a natural language as they would write when communicating to members of their team.

Basically, RNN based network is used in this to read the input text (i.e., the sequence description text) word by word. The order of the sentence formation is maintained so as to learn the meaning of input text. Each word is processed through a RNN unit which can either be LSTM or GRU. LSTM and GRU are the types of RNN which have their own essential rules. LSTM has the capability of retaining maximum information of long sequences and GRU (Gated Recurrent Unit) has the ability to forget irrelevant information and retain only the important information with regard to the context. These units process all the words one by one and hence generate output information as its resultant. We have also used bidirectional layer which reads the input text form both the directions (i.e., both forward and backward) improving the performance of the model on this sequence classification. As we know that in the field of AI, everything is all about numbers, vectors, matrix and statistics, so we can say that a model can only feed numbers and it can only infer probabilities and the maximum probability is always chosen. Our model follows the same to predict the most probable output. We have also focused on embedding as neural network only accepts numbers rather than string values, that is basically treating each input text word by making their vector forms which ultimately represents each word with some fixed size vector of numbers. Apart from embedding, attention algorithm has also been used which helps in predicting more closely expected outputs, i.e., the most probable expected output sequence. This attention layer helps in giving word or vectors more weightage by giving them scores and comparing them with the output during the training of the model itself. This weightage help in predicting more close expected outputs for desired input sentences. Ultimately everything comes down to the part of dataset on which the model gets trained. The data that is fed to the model for training should be as good as possible ,i.e., the dataset should be apt to expect a greater performance from the model.

We have carefully created our corpus by getting references from actual used register programming sequences in the EDA industry. We have introduced a wide variety of cases including cases with augmented data or noise . This robust model a great accuracy covering all most all scenarios of sequences that can be used by a designer for description of a input text sequence.

Implementation

This model has been deployed and using Django Framework to maintain communication between the model and iDSNG (our spec entry tool) through various APIs handling multiple requests at a single time.

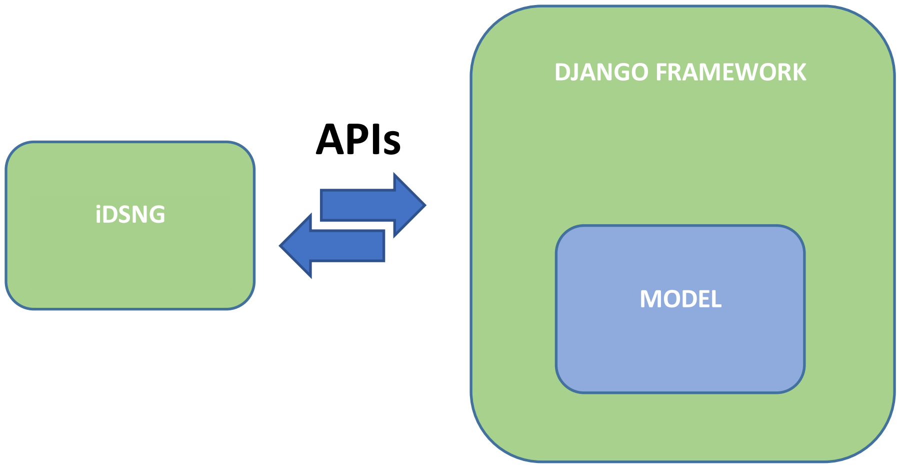
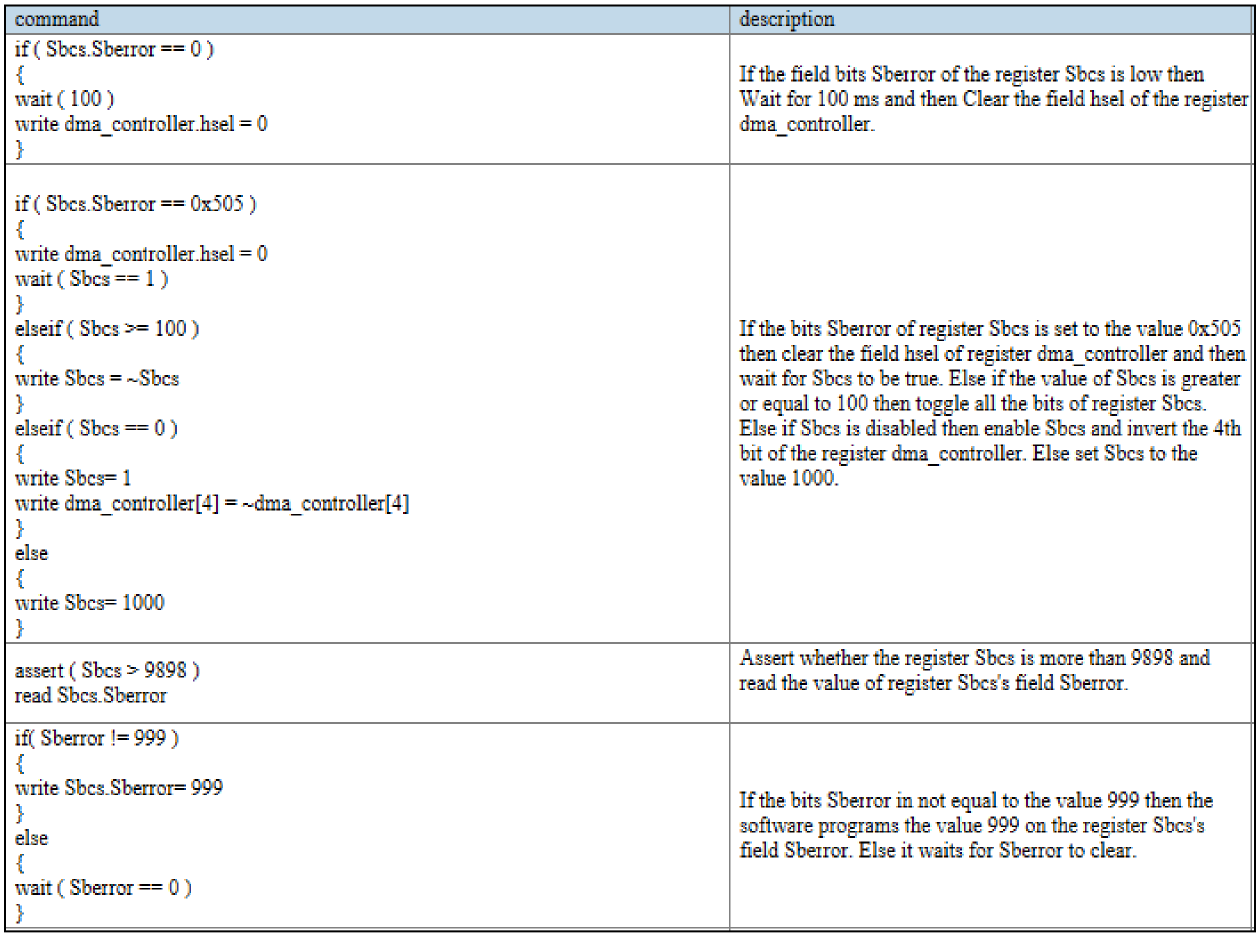


Figure 3 -COMMUNICATION FRAMEWORK

The above communication network represents the interaction of iDSNG with the model through rest APIs.

Below is the figure (Figure 4) depicting the results of the input text, .i.e., the sequence description below the column ‘description’ and its predicted output by the model in the column ‘command’.

The specification used in the example figure (Figure 4) basically consists of the register 'dma_controller' having the field 'hsel' and the register 'Sbcs' having the field 'Sberror' .



command	description

if (Sbcs.Sberror == 0)
{
wait (100)
write dma_controller.hsel = 0
}
	If the field bits Sberror of the register Sbcs is low then Wait for 100 ms and then Clear the field hsel of the register dma_controller.

if (Sbcs.Sberror == 0x505)
{
write dma_controller.hsel = 0
wait (Sbcs == 1)
}
elseif (Sbcs >= 100)
{
write Sbcs = ~Sbcs
}
elseif (Sbcs == 0)
{
write Sbcs= 1
write dma_controller[4] = ~dma_controller[4]
}
else
{
write Sbcs= 1000
}
	If the bits Sberror of register Sbcs is set to the value 0x505 then clear the field hsel of register dma_controller and then wait for Sbcs to be true. Else if the value of Sbcs is greater or equal to 100 then toggle all the bits of register Sbcs. Else if Sbcs is disabled then enable Sbcs and invert the 4th bit of the register dma_controller. Else set Sbcs to the value 1000.

assert (Sbcs > 9898) read Sbcs.Sberror	Assert whether the register Sbcs is more than 9898 and read the value of register Sbcs's field Sberror.

if(Sberror != 999)
{
write Sbcs.Sberror= 999
}
else
{
wait (Sberror == 0)
}
	If the bits Sberror in not equal to the value 999 then the software programs the value 999 on the register Sbcs's field Sberror. Else it waits for Sberror to clear.

Figure 4- Description of sequence and the predicted output sequence

LIMITATIONS

Issue remains with words which are unknown in the vocabulary/dictionary of the model. For such words model may produce unexpected outputs as those are out of context for the model.

In some cases, failure in data interpretation occurs because of insufficient data input.

The model may produce unexpected results if not trained with enough data that is accurate, i.e., the training dataset needs to be large and accurate.

Computational power, inference time, etc. comes in as hindrance for viable usage with larger architecture.

CONCLUSION

We have been able to handle a wide variety of cases with an accuracy as good as more than 90% with no delay in inference time. This model can effectively handle noise in the input text and thus give attention to only the relevant part of the text that has any influence in the output sequence generating correct output sequences .We have achieved this by improving on and working around the limitations like sufficient and correct amount of dataset to train the model, improving the inference time by reducing the complexity of the model architecture and other issues that we faced along.

REFERENCES

1. https://www.agnisys.com/ids_nextgen/
2. https://content.riscv.org/wp-content/uploads/2018/05/15.55-16-30-UL-001906-PT-C-RISCV-Debug-Specification-Update-and-Tutorial-for-Barcelona-Workshop.pdf
3. https://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-vol-3b-part-2-manual.pdf
4. https://www.analog.com/media/en/technical-documentation/user-guides/ADV7511_Programming_Guide.pdf