Autonomous Detection of Regular Languages

Joel Sommer, Lingjing Huang

# Objective

The objective of this project is to determine whether a neural network trained on strings that are in a regular language is able to determine if a string is in said language. Ideally, the neural network would be capable of analyzing an arbitrarily long string.

# Performance Requirements

The performance requirements have changed from what was originally planned. At first, it was believed that performance will be evaluated by using a Receiver Operating Characteristics ROC curve developed from the metrics collected through training and testing the neural network. After spending time performing the training and assessment of the neural network it was determined that the binary accuracy metric of the trained model was a sufficient measure of performance.

# Background

While, by definition, a regular language does not require any sort of memory construct such as a stack or queue to process, it is believed that a Recurrent Neural Network (RNN) will be required to adequately determine if a string is in the language. This is because the network will be required to learn the states and transition criteria of the regular language, that is, the network will have to have some awareness of the states that came before.

# Source Software

1. Libraries: Keras and TensorFlow are the two primary neural network frameworks to be used. Other libraries such as numpy, matplotlib will also be used data manipulation and visualization.
2. Programming Language: Python. Python was the first client language supported by TensorFlow and currently supports the most features.

# Experimental Plan

1. The project plan is to have at least 2 parts, train a neural network on an input dataset that is composed of an equal number of valid and invalid strings of varying length.
2. Use LSTM Networks Model. Long Short-Term Memory networks – usually just called “LSTMs” – are a special kind of RNN, capable of learning long-term dependencies.
3. Data input: At least 1000 valid strings and 6000 invalid strings for the training.

# Neural Network Model

Through researching the RNN models available in TensorFlow, it was found that the CuDNNLSTM model was the best candidate because 1) it is based on the LSTM cell and 2) it supports very good hardware acceleration which will reduce the training time. Key to the model is the LSTM cell, which is represented in Figure 1.

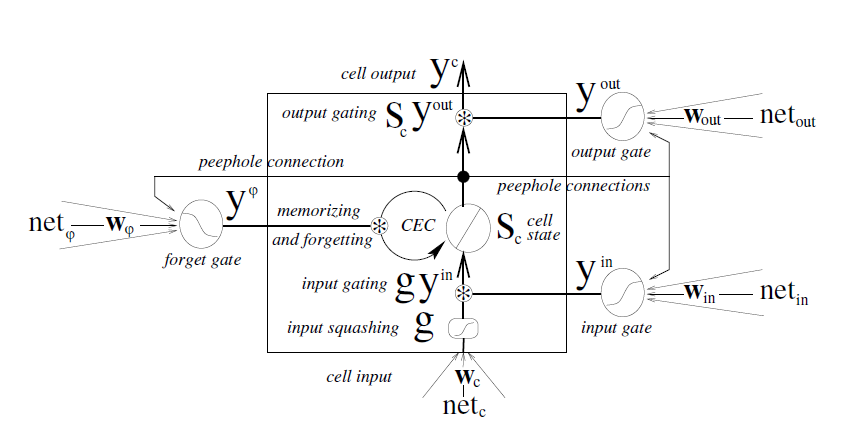


Figure 1: The language models – one cell of the LSTM model [1]

This cell design is from the work done in [1]. The authors of [1] learned that it was important to have “forget gates” or order to provide a mechanism to rest a given cell’s state and improve overall performance of the network. We are using LSTM with forget gate and recently introduced peephole connections.

Going back to the cell in Figure 1, the basic unit of the LSTM network is the memory block containing one or more memory cells and three adaptive, multiplicative gating units shared by all cells in the block.

For the input

1. Input gate activation
2. Forget gate activation
3. Cell input and cell state

Output gate activation and cell out

This approach differs from the approach described in [2] and [3] in that it allows for the ability to perform something akin to a reset. This is important because the approaches described in [2] and [3] are, more or less, to sample all of the data in given data set and predict the next word that may be from the context of all that has been before. The concern is that with this probabilistic style may make sense from a signal processing perspective but does not necessarily apply from a state machine perspective. The rules for the two are not necessarily congruous.

# Datasets

Initially, it was planned to assess multiple regular languages during this experiment. However, it was found that the fundamental object was not able to be met for a single DFA and so the dataset was groomed around a single DFA in an attempt to conserve efforts. The final datasets produced are based on DFA 0, described in Figure 2, below.

The training data sets produced included every string from length 4-32 that are in DFA 0 and every string that is not in DFA 0. The testing data set contains a range of string lengths from 4 to 991 with multiple samples therein. In total, there are 98577 valid test samples and 99001 invalid test samples.

图片包含 剪贴画

已生成高可信度的说明

Figure 2: DFA 0

# Proof of Concept

The proof of concept showed that when DFA 0 was trained with a fixed input string length of 32 symbols using 10,000 samples that were in the DFA and 20,000 samples that were out of the DFA, the model hand 91% accuracy. In this proof of concept, the NN was trained on a total of 30,000 samples and then presented with a new set of 30,000 samples.

During the proof of concept, the NN was composed of an CuDDNLSTM cell that accepted an input of shape (32, 1) and 1 output unit. The thinking is that the CuDDNLSTM would evaluated one sample of the 32-long array of symbols at a time and summarize this information an either a 1 or 0 for in or out of the language. In retrospect, the way the input is fed into the CuDDNLST cell may not be accurate.

After the CuDDNLSTM cell, a “tanh” activation layer was employed with the belief that it would help clamp the output of the CuDNNLSTM cell as either “in” or “out”. It appears this works, but there is little to no grace during the training process. That is, if the NN’s trainable parameters happened to train “correctly”, they would stabilize with high accuracy. An incorrect initial training would never recover.

The “tanh” activation layer was later adjusted to use a sigmoid function as it functioned more reliably.

# Comprehensive Testing

After the proof of concept phase was carried out, the core training and testing code was updated to allow for variable length samples to be fed into the NN. The current implementation allows for a maximum length to be pre-defined. Input symbols of 0 (zero) were arbitrarily selected to represent a non-symbol. Input symbols from the already generated data files were transposed at the time of reading so that a 1 was added to the symbol and then the value was multiplied by 3. So, zeros became 3s and ones became 6s. These were arbitrary choices.

Also, during this stage, it was realized that the method of loading the samples used during the proof of concept was not loading an equal number of classifications of data which may introduce some biasing. It was also recognized that the data sets were being loaded sequentially. Because of the way the data sets were created, this may also introduce some biasing. Options were added to allow sequential, random, and loading by some even interval in order to assess if there was any impact to the accuracy of the classification. In fact, there was. The randomized input data performed the best, but the gain was very small. Unfortunately, the data from that testing was lost, but it was determined that there was no point in re-running the testing because it would not impact the overall results.

Once training began with the variable length inputs, it was decided that 10-fold cross-validation should be used in order to adopt a more standard form of testing and training. During this stage of training and testing it homogenous length inputs were used to train and test, but the model was capable of accepting variable length inputs. It was found that the maximum accuracy of the results was around 80% for inputs with 5 symbols and close to 85% for inputs with 30 symbols.

At this point, the training/testing philosophy was altered. After evaluating the 10-fold cross-validation it was decided that the method was not providing any significant advantage to the training process. Instead, for the training process, it was decided to train for some very large epoch value but employ an “EarlyStopping” callback which would abort the training once there was no significant improvement in the mean absolute error. The understanding of how to conduct the testing process changed to be one where training was performed on a relatively small sample size and symbol length but would be performed against a large variable length pool of data that likely has not been presented to the model. Here, a testing data set was produced where goal of 1000 valid and invalid string were produce for lengths of 1 to 991 with a step size of 10. Obviously, there are not 1000 strings that can be produced at length 1, so only 1 valid and 1 invalid sample were created.

When training using this new methodology the accuracy performance was, once again, very close to 85%. However, when tested with the large data set of 99,577 valid and 99,001 invalid strings the model only had an accuracy of just under 50%.

# Follow on Questions and Investigations

Admittedly, the understanding of machine learning by team is very limited. However, with the knowledge gained from this project, it is believed that the model can be updated to include more LSTM cells and cross linking of the layers to allow for more awareness of the symbols being feed into the LSTM. Additionally, it is believed that there was a fundamental flaw in the understanding of how the LSTM cell reads data, as discussed in the Proof of Concept section and that given more time some significant gains can be made in the overall performance of the model.

# Final Deliverables

The final deliverables are:

1. The final report – a document, which will summarize our project’s idea, objective, progress, accomplishments and results.
2. Raw data – all samples (images) we generated and/or utilized in the project.
3. Presentation slides ­– brief overview of our project for the final in class presentation.
4. Webpages – contain all information about the project.

# References

|  |  |
| --- | --- |
| [1] | F. A. Gers and E. Schmidhuber, "LSTM recurrent networks learn simple context-free and context-sensitive languages," *IEEE Transactions on Neural Networks,* vol. 12, no. 6, pp. 1333-1340, Nov 2001. |
| [2] | R. Atienza, "LSTM by Example using TensorFlow," 17 Mar 2017. [Online]. Available: https://towardsdatascience.com/lstm-by-example-using-tensorflow-feb0c1968537. |
| [3] | "Recurrent Neural Networks," 27 Jan 2018. [Online]. Available: https://www.tensorflow.org/tutorials/recurrent. |
| [4] | C. Olah, "Understanding LSTM Networks," 27 Aug 2015. [Online]. Available: http://colah.github.io/posts/2015-08-Understanding-LSTMs/. |
| [5] | "language model - extension of RNN," 8 Sep 2017. [Online]. Available: http://www.cnblogs.com/xinbaby829/p/7127451.html. |