Torc White Paper

November 11, 2018

Rani Fields

Abstract

Torc is a format which aims to enable recurrent neural networks to read JavaScript files for classification purposes. Torc is fully open source. A torc file is a special kind of encoding for JavaScript files and is presented to the user as a CSV. When implemented, customization is encouraged but not required; a user can employ torc as-is. Due to pressing needs, torc's current focus is on security. Long-term goals include non-security use cases such as quality assurance and performance analysis.

Preface: Where we stand today

When we peer into the world of JavaScript and recurrent neural networks, or RNNs, we find ourselves constrained to simply running neural networks in JavaScript; there appears to be no standard library which is capable of loading JavaScript into neural networks. This is of special interest since RNNs specialize in time series data and various other classes of sequential data. Thus, logic follows that RNNs are theoretically be well suited to classification tasks in the domain of programming languages, and of interest, interpreted languages such as JavaScript.

Bridging the JavaScript-RNN Gap

Loading JavaScript into RNNs is a non-trivial affair. In order to achieve our singular goal, we need to somehow translate JavaScript into some sequential format with a static number of columns while maintaining as much structural information as possible and minimizing noise. This is a daunting set of requirements but if we employ certain standard code interpretation processes in tandem with a general mindfulness of how RNNs work, we can sufficiently achieve our goal of bringing JavaScript into the realm of RNN use cases.

Introducing Torc

Torc is our open source bid, and perhaps the first major bid of it type, to bridge this gap. Torc is both an interim format and the method used to generate this interim format. If a user were to open a torc file, the user would see a CSV. This is intentional- torc is fundamentally a CSV. Due to the frequency of CSVs in the machine learning world and due to the nature of our problem at hand, the CSV format filled a natural niche in how we present our re-encoded JavaScript file. When we look at torc files in depth, users will find that torc actually encodes a sequence of events alongside special metadata designed to contextualize provided information wherever possible. Through the specially engineered torc format, users can then load their JavaScript files into RNNs as they please.

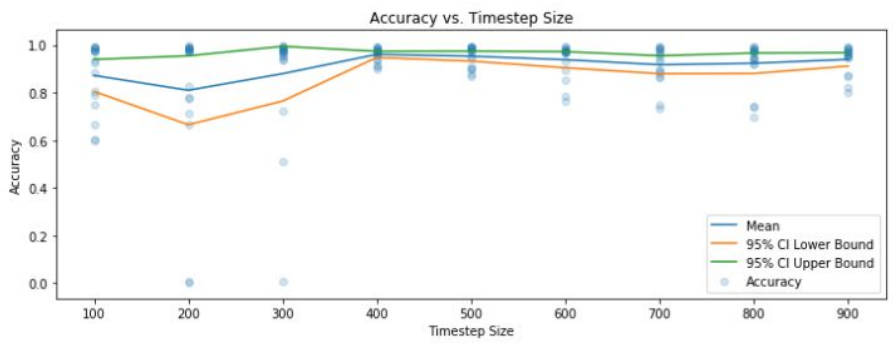
The Torc Method

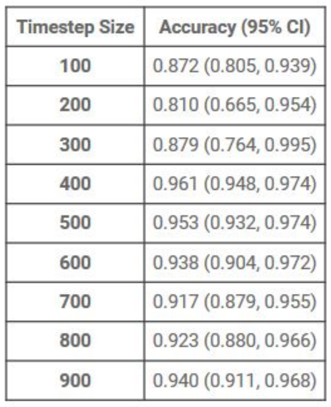
The JavaScript encoding method we found which satisfies the requirements previously outlined is, just like the subject matter, non-trivial. You can read the full methodology in the extended white paper. The core concept is to take to take some JavaScript code, generate a tree based on the structure of the code, perform a specific set of modifications to the tree while collecting metadata on certain data within our tree, then perform a final walk through the tree and record important information as we see it. We then have what is effectively the results of a depth-first search on a de-noised abstract syntax tree with metadata about the different nodes provided where possible. The current metadata and denoising process has been documented in depth in the extended white paper, but the general intention is to provide information about the uniqueness and usage of variables present in our JavaScript code in order to score the relative uniqueness of a variable without explicitly declaring a variable as unique. We also provide metadata focused around other interesting entities such as text.

How Well Does It Work?

In order to test the efficacy of torc, we ran a large number of simulations oriented around building models under different circumstances. The parameters we varied were the number of lines we fed into our RNNs, and the data we fed in (randomly sampled among a pool of 650 good samples, 40k malicious samples). Because our goal was to verify the efficacy of the torc format and not to create functional models, we sampled and resampled our data such that we had an equal number of good and malicious samples. Our final verification dataset was completely exclusive from our train dataset. The model employed was a straight-forward stacked LSTM with a single dense node (sigmoid) as our output. Models varied only by timestep size. No dropout was applied. Please refer to the extended white paper for more information on how we conducted our efficacy checks.

Individual accuracy points by timestep size with accuracy averages and 95% confidence intervals provided below, 17 to 18 samples per group:





Timestep Size	Accuracy (95% CI)
100	0.872 (0.805, 0.939)
200	0.810 (0.665, 0.954)
300	0.879 (0.764, 0.995)
400	0.961 (0.948, 0.974)
500	0.953 (0.932, 0.974)
600	0.938 (0.904, 0.972)
700	0.917 (0.879, 0.955)
800	0.923 (0.880, 0.966)
900	0.940 (0.911, 0.968)

As expected, our data underperforms with fewer timesteps and begins to converge with a certain number of timesteps. With proper tuning, it should be trivial to produce a model capable of handling longer sequences in a more accurate manner.

Torc Use Cases

While the proof of concept revolves around using torc as an intelligent counter-countermeasure, that is, a system designed to detect if a script is attempting to circumvent existing security controls, torc is fundamentally designed to handle a variety of use cases. In general, the use cases of torc will follow existing code analytics use cases. The key determinant in which use cases are available ultimately boils down to the features the implementer employs and whether the implementer opts to create new features. Thus, if torc is implemented with features designed to estimate the likelihood that a piece of code will break, it is theoretically possible to use torc to gauge whether a change in some code segment will result in increased errors in production. Along the same line of thought, torc can theoretically also be used to determine if a code segment is likely to cause performance problems when deployed.

The Future of Torc

Torc has a few paths it can develop along. Of note, we can focus on horizontal expansion (onboard new languages) and vertical expansion (expand core offerings and add new features). Under the domain of new languages, torc is likely to stick to interpreted and command languages due to how tightly torc is coupled with a script's abstract syntax tree. The primary languages we would target are shell and python with shell given special priority. If we were to expand vertically, new features would focus on summarizing functions to determine function uniqueness as well as applying other interesting methods such as pagerank to score the importance of certain variables. We can also explore the ordering of the code itself.

Contact Us

If you have any questions or are interested in getting involved, please reach out to us on github @rafields/torc. Alternatively, you can contact me by email at torc2019@gmail.com.