Programmers spent 50% of their effort in finding and fixing software bugs\([1]\). Software bug detection becomes more challenging due to prevalence of diverse platforms (e.g., multicore, many cores, data centers, accelerators etc.). Cassandra uses neuromorphic hardware for invarient based software bug detection. Current schemes extract correct invariants of a program by analyzing many execution traces. The invariants are stored along with the program. At runtime, as each invariant occurs, it is checked against the stored invariants to determine if it is incorrect (and hence, buggy). Cassandra uses hardware based neural networks to perform online testing and training of invariants alternatively during a program execution. As a result, Cassandra can adapt to new inputs, execution environments, and even code with less execution overhead.

**Introduction**

**How It Works?**

**Offline Training**
- Profiling RAW(Read After Write) dependences from some correct executions.
- Extracted RAW dependences are program invariants.
- Train Neural Network with seq. of RAW as input in a sliding window.

**Online Testing**
- The neural network calculates its output for the RAW dependence sequence formed with last N RAW(S1).
- If(N) each load(I) instruction of shared data and last store instruction(S) of the shared data forms(S2).
- If the output is positive, the sequence is valid. Otherwise, the sequence is invalid.
- Invalid sequences (i.e., instruction addresses) along with the neural network output is recorded into Debug Buffer.

**Online Training**
- If misprediction rate is above certain threshold, Cassandra enters into Online Training mode.

**Neural Network in Hardware**
- A fully configurable neural network time multiplexes an arbitrary network topology to a fixed number of neurons and incurs scheduling overhead.
- We propose a partially configurable neural network with only one hidden layer. We limit the maximum number of inputs to a neuron to M (N to N).
- Requires only one output(Nu) from the neural network.
- Use three stage pipeline without time multiplexing – S1(input), S2(hidden layer) and S3(output).
- Each of S2 and S3 takes T cycles where T is the number of cycles it takes for a neuron to calculate the output.
- If the FIFO is full, this pipeline takes an input(S1) after every T cycles.

**Offline Pruning and Ranking**
- The Debug Buffer contains last few (e.g., 600) invalid RAW dependence sequences.
- Process the contents of the buffer in two steps - Pruning and Ranking. The post processing is done offline after a failure.
- Pruning: Remove all the entries in the Debug Buffer that matches with Correct Set (inputs used for offline training or from input generator using any correct executions).
- Ranking: Ranking algorithm produces higher rank for a sequence if it has more matched dependences (comparing with Correct Set).
- More RAW dependences to match indicates that the mismatched dependence is closer to the root cause of the bug.

**Motivation**

- Data communication(RAW) dependences patterns are different in correct and buggy executions.
- Neural Networks are efficient in learning patterns.
- Neural Networks are becoming popular as an alternative accelerator for different purposes.
- Can detect some unseen bugs after learning communication pattern.

**References**

1. Tom Britton, Lise Jeng, Grzegorz Carmon, Paul Chekal, and Tamer Catteelablage. Reversible debugging software