

Ensuring Safety in Deep Reinforcement Learning for Systems

The whiRL Approach

Akhoury Shauryam
akhoury@cmi.ac.in

Chennai Mathematical Institute

October 31, 2023

Table of Contents

- 1 Objectives
- 2 Why Verify RL?
- 3 Aurora
- 4 Properties
- 5 Safety vs Liveness
- 6 Model Checking through BMC
- 7 BMC for DRL
- 8 Results

Objectives



Why Verify RL?

- Deep Reinforcement Learning (DRL) has seen a remarkable surge in its application within the domains of computer and networked systems in recent years. This surge is largely driven by the promise of DRL to make intelligent decisions in complex, dynamic environments.
- However, as DRL policies make decisions, they often do so in ways that are challenging to understand and interpret. The decision-making process is obscured within complex neural networks. This obscurity raises a fundamental question: Can we trust these policies to make safe decisions?

How is it different to DNN Verification?

Why Verify RL?

- **Single Invocation:** DNN verification tools typically focus on a single invocation of the DNN. In DRL, where DNNs are invoked repeatedly and their behavior evolves over time, this limitation is restrictive.
- **Scalability:** The NP-complete nature of DNN verification results in exponential worst-case complexity, making DRL verification via DNN-style approaches highly challenging.

Opportunities

Why Verify RL?

- Formal verification offers a systematic, mathematical approach to assess whether a given system, in this case, DRL policies, satisfies predefined requirements or exposes specific vulnerabilities.
- In this presentation, we will delve into the details of how formal verification, through the whiRL platform, can help us ensure the safety and reliability of learning-augmented systems, and we'll illustrate its application in real-world scenarios.

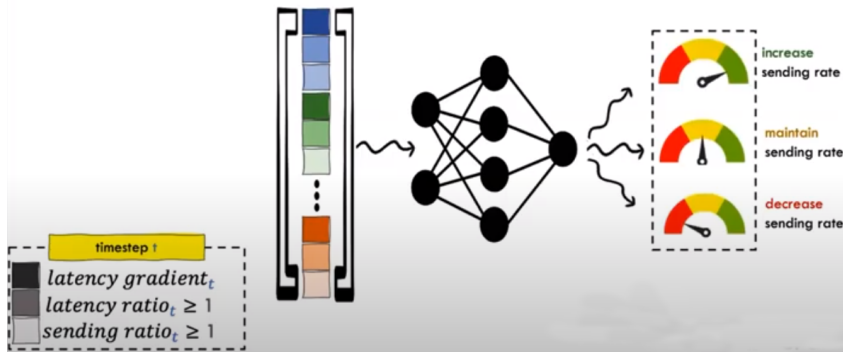
- Aurora is a DRL-based Internet congestion control algorithm designed to optimize network traffic and reduce congestion.
- It employs a DNN to make decisions about adjusting the sending rate based on observed network statistics.
- It dynamically adjusts the sending rate in response to changes in network conditions, aiming to optimize throughput and minimize latency.

The DNN takes a specific input vector with $3t$ entries, where t is a fixed positive integer. This vector includes three sets of t entries:



- **Latency Ratios:** t entries representing the observed latency ratios, which are the ratios between experienced latency and the minimum latency observed so far.
- **Sending Ratios:** t entries representing the observed sending ratios, which are the ratios between the number of packets sent and the number of packets arriving at the destination.
- **Latency Gradients:** t entries representing the observed latency gradients, indicating whether the observed latency is increasing or decreasing.

The DNN's single output provides a decision regarding the sending rate, which can be one of the following: Increase, Decrease or Maintain.

The Aurora Congestion Controller



Aurora - Properties

	 Network Conditions	 Desired Output
Property 1	excellent 😊	eventual change 👍👎
Property 2	excellent 😊	eventual increase 👍
Property 3	poor 😞	next-step decrease 👎
Property 4	poor 😞	eventual decrease 👎

- **Property 1 (Liveness):** DNN should eventually change sending rate when past observations indicate excellent network conditions (low latency, no loss).
- **Property 2 (Liveness):** DNN should eventually increase sending rate when past observations reflect excellent network conditions, and DNN output is not positive.
- **Property 3 (Safety):** DNN should decrease sending rate when experiencing high packet loss on a low-latency link, and DNN output is not negative.
- **Property 4 (Liveness):** DNN should eventually decrease sending rate when experiencing high packet loss on a low-latency link, and DNN output is not negative.

Safety

- Safety properties concern the absence of undesirable states or events in a system.
- A violation of a safety property is demonstrated by a finite trace, meaning that a finite sequence of actions or events leads to the undesirable state or condition.

Liveness

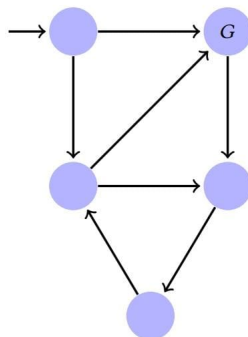
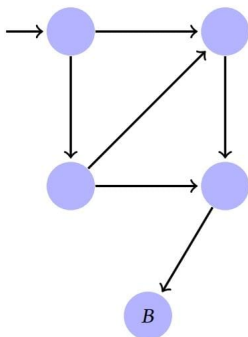
- Liveness properties relate to the occurrence of desirable events or the persistence of a certain behavior in a system.
- A violation of a liveness property is typically demonstrated by a lasso-shaped infinite trace, which indicates that the system gets stuck in a loop or fails to progress towards a desired outcome.

Safety vs Liveness

Visualisation

Safety concerns the prevention of reaching an undesirable state within a transition graph, starting from an initial state. In contrast, Liveness focuses on avoiding getting trapped in a repetitive loop without ever reaching a favorable state.

B denotes a bad state while G means a good state.



How do we go about verifying that our properties hold? One way is to employ BMC to turn it into a SAT-Solving query.

BMC (Bounded Model Checking) is a formal verification technique that examines finite-state systems to find potential errors within a specific number of state transitions. The Inputs to the BMC are:

- S : The State Space
- I : The Initial Predicate
- T : The Transitional Relation, $T(x, y)$ is False if y is not reachable from x
- B/G : Subset of S denoting Bad or Good States

Validity is ensured by taking the Negation and looking for an UNSAT result.

Safety Encoding

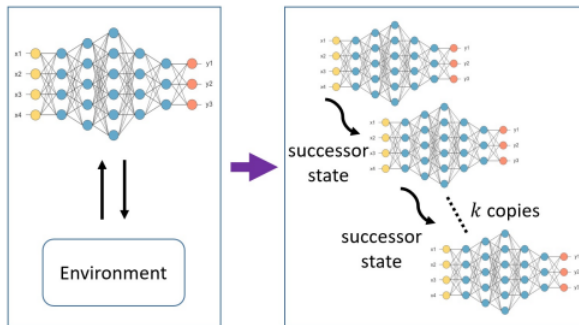
$$\exists x_1, \dots, x_k \in S \mid I(x_1) \wedge \left(\bigwedge_{i=1}^{k-1} T(x_i, x_{i+1}) \right) \wedge \left(\bigwedge_{i=1}^k B(x_i) \right)$$

Liveness Encoding

$$\exists x_1, \dots, x_k \in S \mid I(x_1) \wedge \left(\bigwedge_{i=1}^{k-1} T(x_i, x_{i+1}) \right) \wedge \left(\bigwedge_{i=1}^k \neg G(x_i) \right) \wedge \left(\bigvee_{i=1}^{k-1} x_k = x_i \right)$$

Employing BMC in a DNN

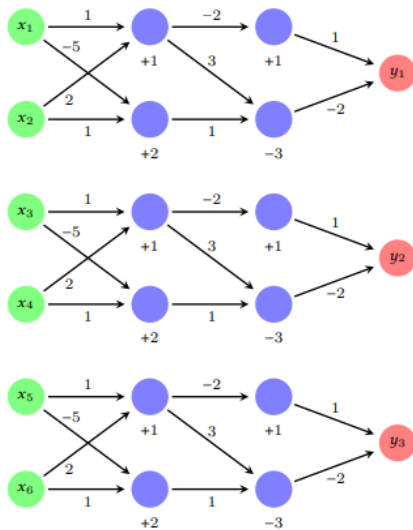
To evaluate the safety or liveness of a system, a technique involves replicating the original input DNN N , k times, creating a larger network, N' . This expansion also extends to the input layer, effectively encoding k successive states.



Then use this N' along with Pre and Post conditions P and Q .

BMC for DNN

Example

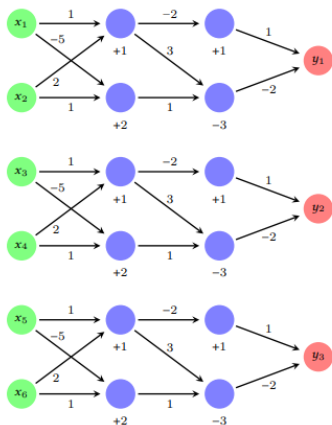


The top-most DNN is our original DNN, suppose for $k = 3$ we want to verify some property.

We copy the DNN thrice and this expanded network has 6 input neurons and 3 output neurons (tripled from the original 2 inputs and 1 output).

BMC for DNN

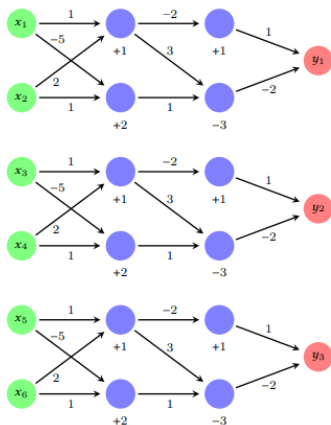
Example Cont.



Through this DNN, we make a relation between (x_1, x_2) and (x_3, x_4) . The DNN returns y_1 , which corresponds to an action. Using the action y_1 , the environment gives us a new state (x_3, x_4) . So we set $T((x_1, x_2), (x_3, x_4))$ as True here. Set T accordingly.

BMC for DNN

Example Cont.

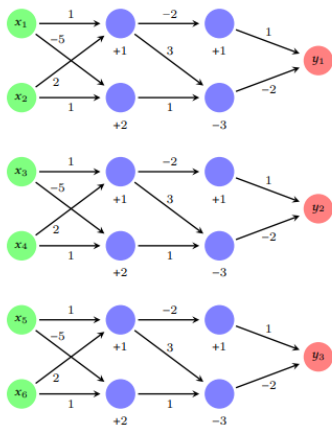


Property P defines constraints on the input neurons:

- They must remain within the range of -1 to 1
- Their increase is limited to at most $1/2$ if the previous output was positive
- Their decrease is restricted to at most $1/2$ if the previous output was non-positive

BMC for DNN

Example Cont.

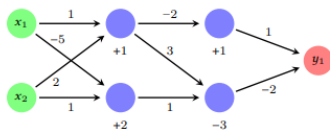


Let $z_i = (x_{2 \cdot i - 1}, x_{2 \cdot i})$ a vector for the same DNN input, then:

$$P = \bigwedge_{i=1}^6 (-1 \leq x_i \leq 1) \\ \wedge \bigwedge_{i=1}^2 (y_i > 0 \rightarrow z_{i+1} \geq z_i + \frac{1}{2} \geq z_i) \\ \wedge \bigwedge_{i=1}^2 (y_i \leq 0 \rightarrow z_{i+1} \leq z_i - \frac{1}{2} \leq z_i)$$

BMC for DNN

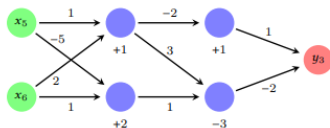
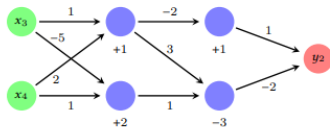
Example Cont.



P along with some Q for Post Condition, say

$$Q = \bigvee y_i \geq 10$$

Passing these P, Q with the extended network, a SAT result will constitute a counterexample, whereas an UNSAT will indicate the absence of any counterexamples.



Summaries of Example

Give results of Aurora, Pensieve Video Streamer, DeepRM

Limitations of whiRL

Does not work with unbounded properties, undeterministic DRL.

Insert Conclusions

Acknowledgment

Insert Image Links, paper link.

Thank You for Your Attention!