Ensuring Safety in Deep Reinforcement Learning for Systems

The whiRL Approach

Akhoury Shauryam
akhoury@cmi.ac.in

Chennai Mathematical Institute

November 2, 2023

Table of Contents

Objectives

- Address the challenge of obscurity in decision-making by DRL policies, which hinders the assessment of the safety of learning-augmented systems for real-world deployment.

Objectives

- Address the challenge of obscurity in decision-making by DRL policies, which hinders the assessment of the safety of learning-augmented systems for real-world deployment.
- Identify an opportunity to use formal verification to establish that a given system meets designer/user-specified requirements or to reveal concrete counter-examples in the context of DRL in systems.

Objectives

- Address the challenge of obscurity in decision-making by DRL policies, which hinders the assessment of the safety of learning-augmented systems for real-world deployment.
- Identify an opportunity to use formal verification to establish that a given system meets designer/user-specified requirements or to reveal concrete counter-examples in the context of DRL in systems.
- Introduce **whiRL**, a platform designed for verifying DRL policies in systems, leveraging recent advancements in deep neural network verification and scalable model checking techniques.

Objectives

- Address the challenge of obscurity in decision-making by DRL policies, which hinders the assessment of the safety of learning-augmented systems for real-world deployment.
- Identify an opportunity to use formal verification to establish that a given system meets designer/user-specified requirements or to reveal concrete counter-examples in the context of DRL in systems.
- Introduce **whiRL**, a platform designed for verifying DRL policies in systems, leveraging recent advancements in deep neural network verification and scalable model checking techniques.
- Demonstrate the utility of **whiRL** by applying it to verify natural requirements in learning-augmented systems in real-world environments like Internet congestion control, adaptive video streaming, and job scheduling in compute clusters.

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.

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?

- **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.

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.

- 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.

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

A DRL Congestion Controller

- Aurora is a DRL-based Internet congestion control algorithm designed to optimize network traffic and reduce congestion.

Aurora

A DRL Congestion Controller

- 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.

- 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:

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.

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.

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.

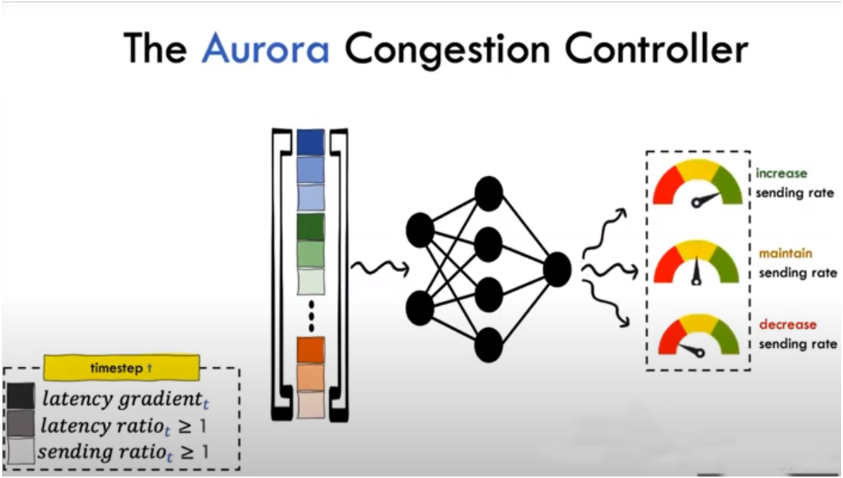
The DNN takes a specific input vector with $3t$ entries, where t is a fixed positive integer. This vector includes three sets of t entries:

- **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.

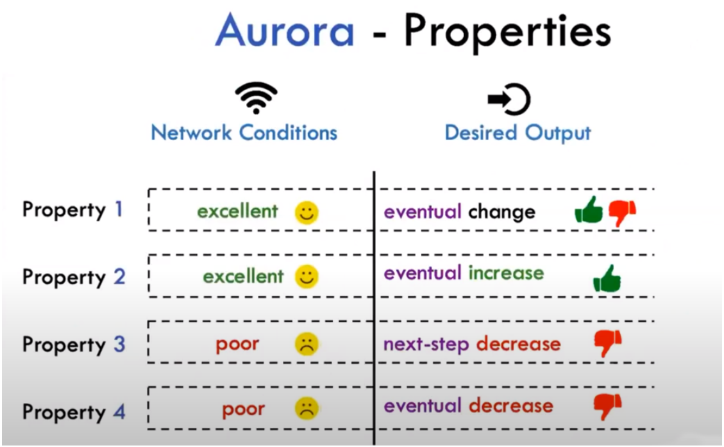
The DNN's single output provides a decision regarding the sending rate, which can be one of the following: Increase, Decrease or Maintain.

Aurora

Architecture Diagram



Properties for Aurora



Properties for Aurora
Formalization

- **Property 1 (Liveness):** DNN should eventually change sending rate when past observations indicate excellent network conditions (low latency, no loss).

Properties for Aurora

Formalization

- **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.

Properties for Aurora

Formalization

- **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 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 vs Liveness

Safety

Safety vs Liveness

Safety

- Safety properties concern the absence of undesirable states or events in a system.

Safety vs Liveness

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.

Safety vs Liveness

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

Safety vs Liveness

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.

Safety vs Liveness

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.

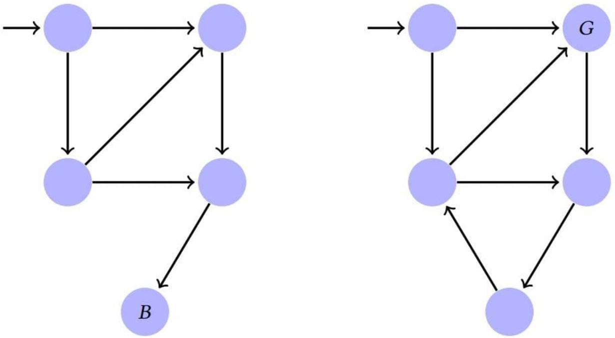
Liveness

- 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.

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:

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

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

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

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
- B/G: Subset of S denoting Bad or Good States

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
- B/G: Subset of S denoting Bad or Good States

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

Setting the Transition system

Assigning values to T depends on us, but we need to adhere to some requirements to fulfil our needs:

Setting the Transition system

Assigning values to T depends on us, but we need to adhere to some requirements to fulfil our needs:

- **Precision Control:** Define $T(s, s')$ for versatile over-approximation, ensuring truth even when s' is unreachable.

Setting the Transition system

Assigning values to T depends on us, but we need to adhere to some requirements to fulfil our needs:

- **Precision Control:** Define $T(s, s')$ for versatile over-approximation, ensuring truth even when s' is unreachable.
- **Under-Approximation Efficiency:** Optimize by setting $T(s, s')$ to false for some reachable states, reducing computational complexity.

Setting the Transition system

Assigning values to T depends on us, but we need to adhere to some requirements to fulfil our needs:

- **Precision Control:** Define $T(s, s')$ for versatile over-approximation, ensuring truth even when s' is unreachable.
- **Under-Approximation Efficiency:** Optimize by setting $T(s, s')$ to false for some reachable states, reducing computational complexity.
- **Avoiding Vacuity:** Carefully prevent setting $T(s, s')$ to false for all states, preventing vacuous system results.

Negation of Safety Encoding

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

Negation of Safety Encoding

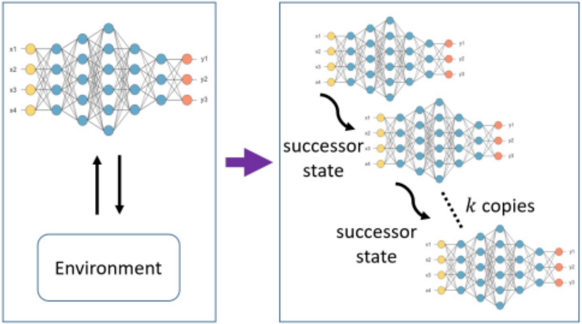
$$\exists x_1, \ldots, x_k \in S \mid I(x_1) \wedge \left(\bigwedge_{i=1}^{k-1} T(x_i, x_{i+1}) \right) \wedge \left(\bigvee_{i=1}^{k} B(x_i) \right)$$

Negation of Liveness Encoding

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

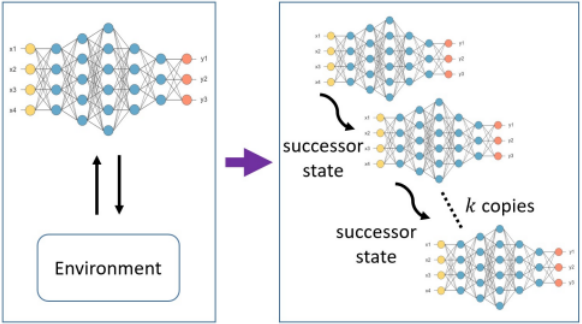
Employing BMC in a DRL

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.



Employing BMC in a DRL

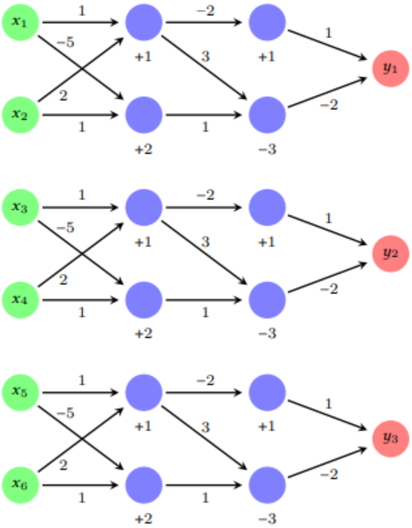
To evaluate the safety or liveness of a system, a technique involves replicating the original input DNN N, k times, creating a larger network, N'. This expansion also extends to the input layer, effectively encoding k successive states.



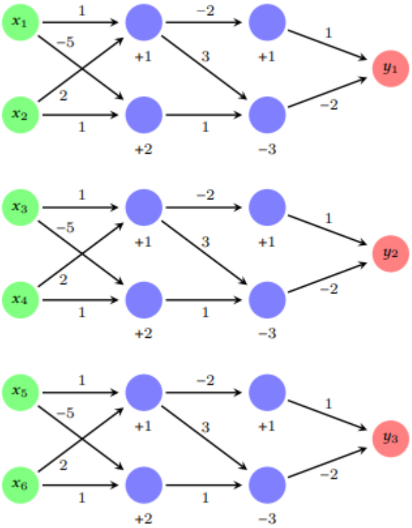
Then use this N' along with Pre and Post conditions P and Q.

BMC for DRL

Example

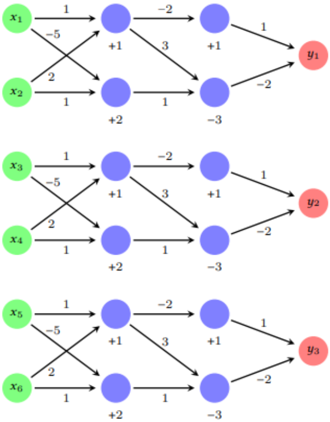


The top-most DNN is our original DNN, suppose for $k = 3$ we want to verify some property.

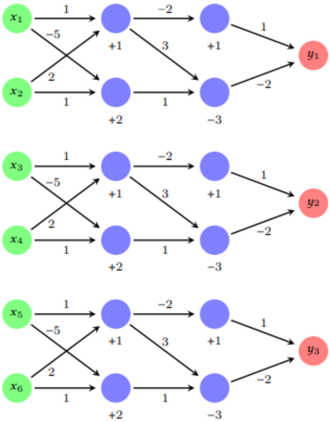


The top-most DNN is our original DNN, suppose for $k = 3$ we want to verify some property.

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).

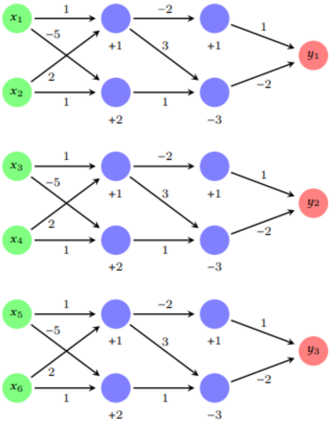


Let s_1 be (x_1, x_2) the first state, and s_2 be the state (x_3, x_4). Here, the DNN N, gives us an action $y_1 = N(s_1)$



Let s_1 be (x_1, x_2) the first state, and s_2 be the state (x_3, x_4). Here, the DNN N, gives us an action $y_1 = N(s_1)$

When using this y_1 in the environment, the environment reacts and transitions our state to s_2.



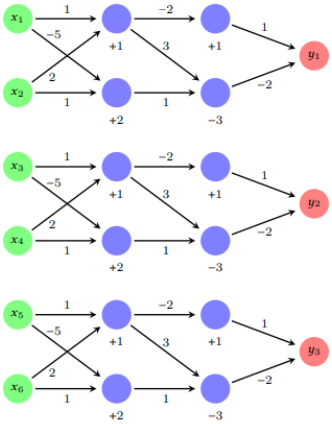
Let s_1 be (x_1, x_2) the first state, and s_2 be the state (x_3, x_4). Here, the DNN N, gives us an action $y_1 = N(s_1)$

When using this y_1 in the environment, the environment reacts and transitions our state to s_2.

Let T_{BMC} be the transition relation here. $T_{BMC}(s, s')$ is False, if s' is not reachable from s.

BMC for DRL

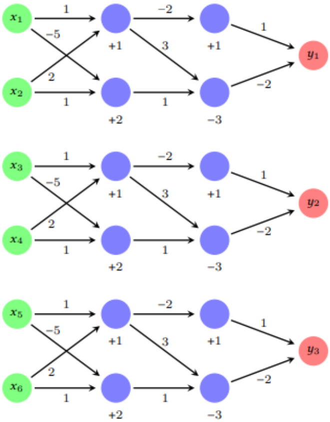
Example Cont.



T_{BMC} here is the transition relation for the unwinded DNN, and through N, $N(s_1) = y_1$ and using y_1 gives us a new state s_2, so we set $T_{BMC}(s_1, s_2) = \text{True}$

BMC for DRL

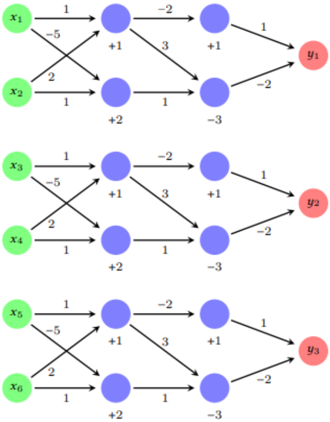
Example Cont.



T_{BMC} here is the transition relation for the unwinded DNN, and through N, $N(s_1) = y_1$ and using y_1 gives us a new state s_2, so we set $T_{BMC}(s_1, s_2) = \text{True}$
So $T_{BMC}((x_1, x_2), (x_3, x_4))$ is set to True for the two states.

BMC for DRL

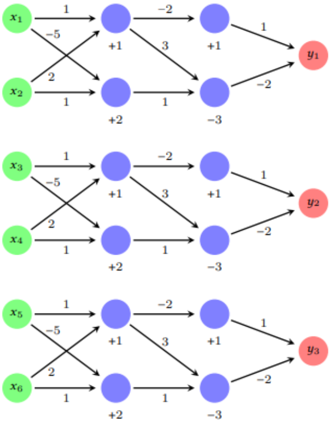
Example Cont.



T_{BMC} here is the transition relation for the unwinded DNN, and through N, $N(s_1) = y_1$ and using y_1 gives us a new state s_2, so we set $T_{BMC}(s_1, s_2) = \text{True}$

So $T_{BMC}((x_1, x_2), (x_3, x_4))$ is set to True for the two states.

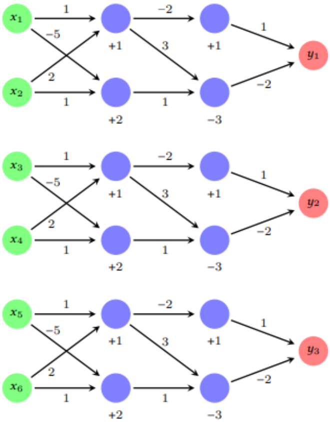
Set T_{BMC} accordingly to complete the transitional relation.



The intial state here is defined as:

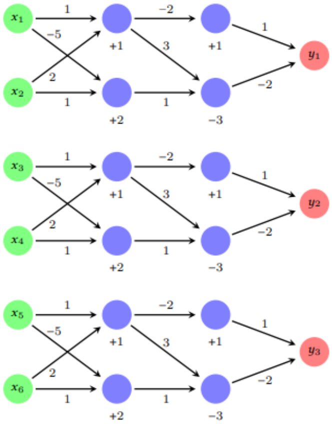
BMC for DRL

Example Cont.



The intial state here is defined as:

- The input must be from the range of -1 to 1

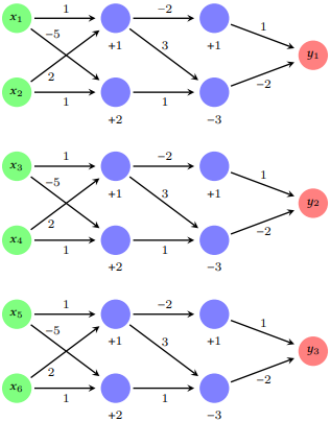


The intial state here is defined as:

- The input must be from the range of -1 to 1

The transitional relation from the enviroment here is defined as:

BMC for DRL
Example Cont.



The intial state here is defined as:

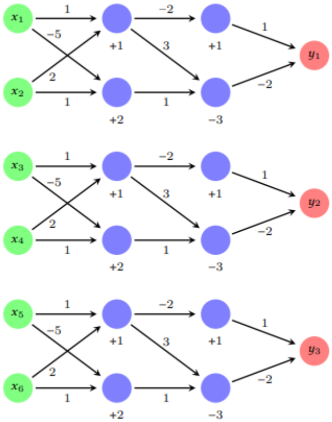
- The input must be from the range of -1 to 1

The transitional relation from the enviroment here is defined as:

- Input's increase is limited to at most 1/2 if the previous output was positive

BMC for DRL

Example Cont.

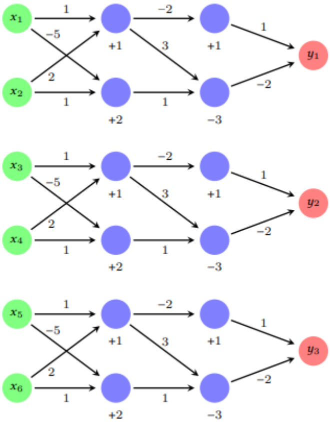


The intial state here is defined as:

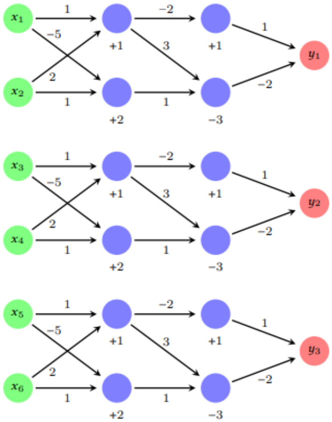
- The input must be from the range of -1 to 1

The transitional relation from the enviroment here is defined as:

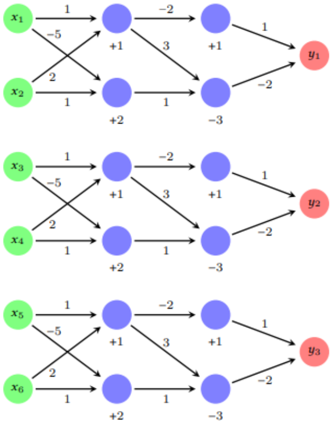
- Input's increase is limited to at most 1/2 if the previous output was positive
- Input's decrease is restricted to at most 1/2 if the previous output was non-positive



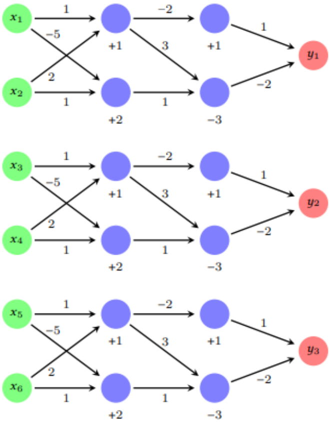
Let $z_i = (x_{2 \cdot i-1}, x_{2 \cdot i})$ a vector for the same DNN input, then:



Let $z_i = (x_{2 \cdot i-1}, x_{2 \cdot i})$ a vector for the same DNN input, then:
$\mathrm{Init}(z_1) = \bigwedge_{i=1}^{2}(-1 \leq x_i \leq 1)$



Let $z_i = (x_{2\cdot i-1}, x_{2\cdot i})$ a vector for the same DNN input, then:
$\text{Init}(z_1) = \bigwedge_{i=1}^{2} (-1 \le x_i \le 1)$
And the transition relation is given by:



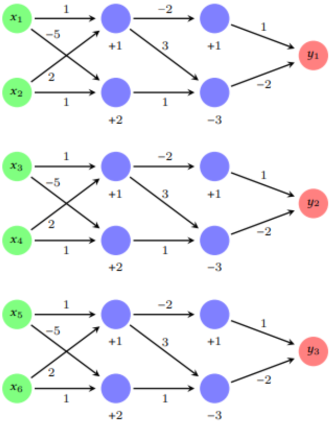
Let $z_i = (x_{2\cdot i-1}, x_{2\cdot i})$ a vector for the same DNN input, then:
$\text{Init}(z_1) = \bigwedge_{i=1}^{2}(-1 \leq x_i \leq 1)$
And the transition relation is given by:

$$T_{BMC}(z_i, z_{i+1}) =$$

$$(y_i > 0 \rightarrow z_{i+1} \geq z_i + \frac{1}{2} \geq z_i)$$

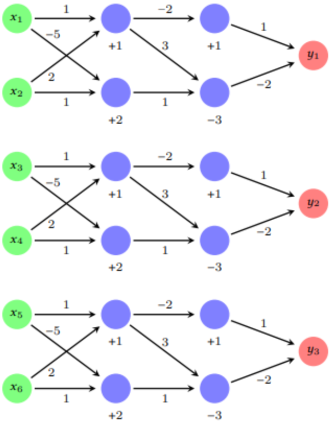
$$\vee$$

$$(y_i \leq 0 \rightarrow z_{i+1} \leq z_i - \frac{1}{2} \leq z_i)$$



Let our Safety Property be that "No output crosses 10". So it can be encoded as

$$\bigwedge y_i < 10$$

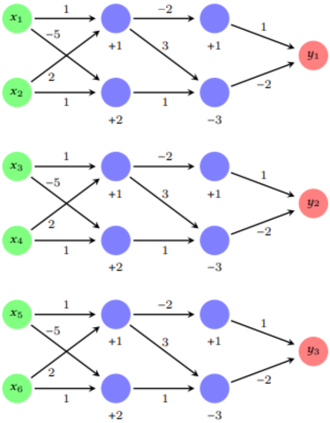


Let our Safety Property be that "No output crosses 10". So it can be encoded as

$$\bigwedge y_i < 10$$

But since we want to check for validity, we'll use the negation of this property,

$$B = \bigvee y_i \geq 10$$



Let our Safety Property be that "No output crosses 10". So it can be encoded as

$$\bigwedge y_i < 10$$

But since we want to check for validity, we'll use the negation of this property,

$$B = \bigvee y_i \geq 10$$

Pass Init, T_{BMC}, B to a SAT-Solver to look if our safety property is verified or if we find a counterexample

How does whiRL use BMC

whiRL takes in 6 inputs:

How does whiRL use BMC

whiRL takes in 6 inputs:

- The DRL agent's DNN (TensorFlow format)

How does whiRL use BMC

whiRL takes in 6 inputs:

- The DRL agent's DNN (TensorFlow format)
- The state space S

How does whiRL use BMC

whiRL takes in 6 inputs:

- The DRL agent's DNN (TensorFlow format)
- The state space S
- Initial State

How does whiRL use BMC

whiRL takes in 6 inputs:

- The DRL agent's DNN (TensorFlow format)
- The state space S
- Initial State
- The Transition relation T

How does whiRL use BMC

whiRL takes in 6 inputs:

- The DRL agent's DNN (TensorFlow format)
- The state space S
- Initial State
- The Transition relation T
- Predicate to define the bad states B

How does whiRL use BMC

whiRL takes in 6 inputs:

- The DRL agent's DNN (TensorFlow format)
- The state space S
- Initial State
- The Transition relation T
- Predicate to define the bad states B
- Parameter k for BMC

How does whiRL use BMC

whiRL takes in 6 inputs:

- The DRL agent's DNN (TensorFlow format)
- The state space S
- Initial State
- The Transition relation T
- Predicate to define the bad states B
- Parameter k for BMC

For a safety verification, whiRL takes these inputs, and copies the DNN k times and constructs the BMC query accordingly, then we get UNSAT if system is safe, SAT otherwise.

Summaries of Example

When using our method on Auora, we get such results:

Summaries of Example

When using our method on Auora, we get such results:

- Property 1: Timed out at $k = 11$

Summaries of Example

When using our method on Auora, we get such results:

- Property 1: Timed out at $k = 11$
- Property 2: Counterexample at $k = 2$

When using our method on Auora, we get such results:

- Property 1: Timed out at $k = 11$
- Property 2: Counterexample at $k = 2$
- Property 3: Counterexample at $k = 1$

Summaries of Example

When using our method on Auora, we get such results:

- Property 1: Timed out at $k = 11$
- Property 2: Counterexample at $k = 2$
- Property 3: Counterexample at $k = 1$
- Property 4: Timed out at $k = 9$

Two resource allocation DRLs, The Penesive Video Streamer and The DeepRM Resource Manager, had 2 of their properties encoded and checked through whiRL. The results:

Two resource allocation DRLs, The Penesive Video Streamer and The DeepRM Resource Manager, had 2 of their properties encoded and checked through whiRL. The results:

- Counterexample found at $k = 2$ for Penesive

Two resource allocation DRLs, The Penesive Video Streamer and The DeepRM Resource Manager, had 2 of their properties encoded and checked through whiRL. The results:

- Counterexample found at $k = 2$ for Penesive
- Counterexample found at $k = 1$ for DeepRM

Summaries of Example

Two resource allocation DRLs, The Penesive Video Streamer and The DeepRM Resource Manager, had 2 of their properties encoded and checked through whiRL. The results:

- Counterexample found at $k = 2$ for Penesive
- Counterexample found at $k = 1$ for DeepRM

Both results took just a few seconds.

Verifying Sufficient Training

The proposed framework works in verifying pre-trained DRL models.

Verifying Sufficient Training

The proposed framework works in verifying pre-trained DRL models. But adding whiRL to the model during training, and adding very high penalties for unwanted scenarios. This leads the DRL to learn safe policies quicker.

Verifying Sufficient Training

The proposed framework works in verifying pre-trained DRL models. But adding whiRL to the model during training, and adding very high penalties for unwanted scenarios. This leads the DRL to learn safe policies quicker.
Further examining which properties are verified could aid in reasoning the sufficiency of training.

Limitations of whiRL

- **Focus on Deterministic DRL Policies:** The framework is primarily designed for verifying deterministic DRL policies, which may not cover systems that rely on stochastic policies.

Limitations of whiRL

- **Focus on Deterministic DRL Policies:** The framework is primarily designed for verifying deterministic DRL policies, which may not cover systems that rely on stochastic policies.
- **Restrictions on Neural Network Types:** The discussion is centered on feedforward neural networks, which are common in DRL-based systems, but other types like recurrent neural networks (RNNs) are not covered.

Limitations of whiRL

- **Focus on Deterministic DRL Policies:** The framework is primarily designed for verifying deterministic DRL policies, which may not cover systems that rely on stochastic policies.
- **Restrictions on Neural Network Types:** The discussion is centered on feedforward neural networks, which are common in DRL-based systems, but other types like recurrent neural networks (RNNs) are not covered.
- **Piecewise-Linear Activation Functions:** The framework works with DRL systems that use piecewise-linear activation functions, like ReLU. Systems incorporating different activation functions may pose challenges for formal verification.

Limitations of whiRL

- **Focus on Deterministic DRL Policies:** The framework is primarily designed for verifying deterministic DRL policies, which may not cover systems that rely on stochastic policies.
- **Restrictions on Neural Network Types:** The discussion is centered on feedforward neural networks, which are common in DRL-based systems, but other types like recurrent neural networks (RNNs) are not covered.
- **Piecewise-Linear Activation Functions:** The framework works with DRL systems that use piecewise-linear activation functions, like ReLU. Systems incorporating different activation functions may pose challenges for formal verification.
- **Incomplete Proofs:** The model uses BMC, so we can only check up to a bound depending on our resources and not prove unbounded properties.

Future Work

For Proof Completeness

- Building upon this work, finding an inductive invariant, either manually or inferred automatically, could help us modify it to a complete proof.

Future Work

For Proof Completeness

- Building upon this work, finding an inductive invariant, either manually or inferred automatically, could help us modify it to a complete proof.
- $\text{Prop}(s) \wedge T(s, s') \implies \text{Prop}(s')$ can return False, so we need to make a stronger precondition to check if the property holds.

Future Work

For Proof Completeness

- Building upon this work, finding an inductive invariant, either manually or inferred automatically, could help us modify it to a complete proof.
- $\text{Prop}(s) \wedge T(s, s') \implies \text{Prop}(s')$ can return False, so we need to make a stronger precondition to check if the property holds.
- To do this, we find an $\text{Inv}(s)$ such that $\text{Inv}(s) \wedge T(s, s') \implies \text{Inv}(s')$

Future Work

For Proof Completeness

- Building upon this work, finding an inductive invariant, either manually or inferred automatically, could help us modify it to a complete proof.
- $\text{Prop}(s) \wedge T(s, s') \implies \text{Prop}(s')$ can return False, so we need to make a stronger precondition to check if the property holds.
- To do this, we find an $\text{Inv}(s)$ such that $\text{Inv}(s) \wedge T(s, s') \implies \text{Inv}(s')$
- We find such an Inv, which makes $\text{Prop}(s) \wedge \text{Inv}(s) \wedge T(s, s') \implies \text{Prop}(s')$ True.

Future Work

For Proof Completeness

- Building upon this work, finding an inductive invariant, either manually or inferred automatically, could help us modify it to a complete proof.
- $\text{Prop}(s) \wedge T(s, s') \implies \text{Prop}(s')$ can return False, so we need to make a stronger precondition to check if the property holds.
- To do this, we find an $\text{Inv}(s)$ such that $\text{Inv}(s) \wedge T(s, s') \implies \text{Inv}(s')$
- We find such an Inv, which makes $\text{Prop}(s) \wedge \text{Inv}(s) \wedge T(s, s') \implies \text{Prop}(s')$ True.
- whiRL2.0 uses this technique and finds Inductive Invariants automatically, so it works with unbounded models to show complete proofs and gets to those proofs faster.

Conclusions

- **Ubiquity of DRL:** DRL is increasingly pervasive, but verifying its behavior and decision-making is complex.

Conclusions

- **Ubiquity of DRL:** DRL is increasingly pervasive, but verifying its behavior and decision-making is complex.
- **Verification:** Efforts have been made to simplify the verification process for DRL systems, a critical step to ensure their reliability.

Conclusions

- **Ubiquity of DRL:** DRL is increasingly pervasive, but verifying its behavior and decision-making is complex.
- **Verification:** Efforts have been made to simplify the verification process for DRL systems, a critical step to ensure their reliability.
- **New Training Metric:** The study introduces a novel metric that assesses the sufficiency of training, providing a valuable tool for determining readiness.

Conclusions

- **Ubiquity of DRL:** DRL is increasingly pervasive, but verifying its behavior and decision-making is complex.
- **Verification:** Efforts have been made to simplify the verification process for DRL systems, a critical step to ensure their reliability.
- **New Training Metric:** The study introduces a novel metric that assesses the sufficiency of training, providing a valuable tool for determining readiness.
- **Future Research:** This research lays the groundwork for further exploration in DRL verification and training assessment, with a focus on ongoing improvement and innovation.

Acknowledgment

- Verifying Learning-Augmented Systems
- Images taken from: SIGCOMM'21 Technical Session

Thank You!