NN2EQCDT
Equivalent transformation of feed-forward neural networks as DRL policies into compressed decision trees

Torben Logemann¹  Eric MSP Veith¹

¹Carl von Ossietzky University Oldenburg
Research Group Adversarial Resilience Learning
Oldenburg, Germany

Contact: torben.logemann@uol.de
RL Basics

- Learning systems have achieved remarkable successes.
- Deep Reinforcement Learning (RL) (DRL) at the core of many remarkable successes

- RL involves
  - learning agents
  - with sensors and actuators
  - to achieve specific goals
  - through trial and error

- using different algorithms
- DRL = RL + Deep Neural Networks (DNNs)
- have proven that they are capable of handling complex tasks.

Figure. RL Architecture [1]
RL application fields

- Learning system are applied in various fields:
  - In healthcare to determine the best treatment policy [2].
  - In robotics, RL agents can learn different tasks to reach higher goals [2]
  - DRL is used in autonomous driving because of its strong interaction with the environment [3].
  - In cybersecurity, DRL is used for automatic intrusion detection techniques and defense strategies [4].
  - In power grid DRL is used for voltage stabilization [5]
Motivation

- DRL agents promise true resilience by learning to counter the unknown unknowns.
- Yet no guarantees about their behavior.
- But a necessity for operators, since otherwise no responsibility can be taken.
- Because of potential to significantly threaten the safety the overall system.
- Architecture to provide such guarantees is presented in [6]
Understanding of agent achievements

- In complex environments agents learn complex behaviors
- Understanding currently: Study of effects of learned strategies in terms of impact on the environment
Understanding of agent achievements: Example

- Adversarial Resilience Learning (ARL) attack agents are deployed with the goal of causing voltage band violations [5]
- Explanation extracted by analysis of the impact of attacker actions on victim buses.
- Not deeply interpreted and no guarantees for all situations
- Guarantees important for defender agents with infinite horizon
Goals

- First step for guarantees is transparency to learned strategies of agents
- Idea: Use Decision Trees (DTs) for explanation
  - DTs are transparent and therefore interpretable
  - They can be trained directly (no need for black-box DNN models)
  - But DNNs are better regularized, which increases trainability [7]
- Conflicting goals:
  - Construction of powerful (DRL) learning system
  - (Post-hoc) Explainability with comprehensible model (e.g. DTs)
Contribution

▶ Equivalent transformation of efficient-learnable Feed-Forward DNNs (FF-DNNs) into compressed DTs

▶ NN2EQCDT algorithm heavily relies on equivalence description of DNNs and DTs [8], but still addressed research gaps to better use it for explainability:
  ▶ Transformation algorithm and actual implementation proposed for PyTorch models
  ▶ Exponential growth is addressed by lossless compression
  ▶ Dynamic compression reduces computation time significantly and may reduce inference time
  ▶ Option to directly include invariants for further compression
Input FF-DNN PyTorch model for NN2EQQCDT

```python
nn.Sequential(
    nn.Linear(2, hid, bias=True), nn.ReLU(),
    nn.Linear(hid, hid, bias=True), nn.ReLU(),
    nn.Linear(hid, 1, bias=True)
)
```

Figure 5. Actor model in PyTorch with variable hidden size

▶ For simple example: \( hid = 8 \)
NN2EQQCDT algorithm

1: \( \hat{W} = W_0 \)
2: \( \hat{B} = B_0^T \)
3: \( \text{rules} = \text{calc_rule_terms}(\hat{W}, \hat{B}) \)
4: \( T, \text{new\_SAT\_leaves} = \text{create\_initial\_subtree}(\text{rules}) \)
5: \( \text{set\_hat\_on\_SAT\_nodes}(T, \text{new\_SAT\_leaves}, \hat{W}, \hat{B}) \)
6: \( \textbf{for} \; i = 1, \ldots, n - 1 \; \textbf{do} \)
7: \( \quad \text{SAT\_paths} = \text{get\_SAT\_paths}(T) \)
8: \( \quad \textbf{for} \; \text{SAT\_path} \; \text{in} \; \text{SAT\_paths} \; \textbf{do} \)
9: \( \quad \quad \textbf{for} \; \text{SAT\_path} \; \text{in} \; \text{SAT\_paths} \; \textbf{do} \)
10: \( \quad \quad \quad a = \text{compute\_a\_along}(\text{SAT\_path}) \)
11: \( \quad \quad \quad \text{SAT\_leave} = \text{SAT\_path}[\neg 1] \)
12: \( \quad \quad \quad \hat{W}, \hat{B} = \text{get\_last\_hat\_of\_leave}(T, \text{SAT\_leave}) \)
13: \( \quad \quad \quad \hat{W} = (W_i \odot [(a^T)_x k]) \hat{W} \)
14: \( \quad \quad \quad \hat{B} = (W_i \odot [(a^T)_x k]) \hat{B} + B_i^T \)
15: \( \quad \quad \quad \text{rules} = \text{calc_rule_terms}(\hat{W}, \hat{B}) \)
16: \( \quad \quad \text{new\_SAT\_leaves} = \)
17: \( \quad \quad \text{add\_subtree}(T, \text{SAT\_leave}, \text{rules}, \text{invariants}) \)
18: \( \quad \text{set\_hat\_on\_SAT\_nodes}(T, \text{new\_SAT\_leaves}, \hat{W}, \hat{B}) \)
19: \( \text{convert\_final\_rule\_to\_expr}(T) \)
20: \( \text{compress\_tree}(T) \)

Figure 1. NN2EQQCDT algorithm
Effective weight matrix calculation

Figure 2. Algorithm for calculation of effective weight matrices with right-handed linear transformation and bias for ReLU activation function, based on [15]
XOR model: DT Construction

\[
\hat{W}_0 = \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}
\]

\[-X_0 + X_1 > 0\]

\[
\hat{W}_1,a = \begin{pmatrix} 0 \\ 0 \end{pmatrix}
\hat{W}_1,a = \begin{pmatrix} -1 \\ 1 \end{pmatrix}
\hat{W}_1,a = \begin{pmatrix} 1 \\ -1 \end{pmatrix}
\]

\[Y = 0\] \[Y = -X_0 + X_1\] \[Y = X_0 - X_1\]

Figure. Simple example of an DT representing an XOR function constructed
XOR model: DT Compression

\[ X_0 - X_1 > 0 \]

\[ -X_0 + X_1 > 0 \]

\[ Y = X_0 - X_1 \]

\[ Y = -X_0 + X_1 \]

**Figure.** Simple compression example
Simple example: Model car in MCC
Simple example: Decision tree
Simple example: 3D Plot
Calculation of amount of nodes

- Calculation of amount of nodes of a DT

\[
\#_{\text{nodes}} = \sum_{i=0}^{d-1} 2^i
\]

- according to the equivalence description of [8]
- without compression
- depends on the depth of each layer \( d = \sum_{i=0}^{n-2} m_i \)
- with the number of filters in each layer \( m_i \)
Comparison of construction methods

Table. Comparison of results or calculations for the construction of a DT from the simple model without and with compression of the NN2EQCDT algorithm

<table>
<thead>
<tr>
<th>Compression</th>
<th>#_nodes</th>
<th>Computation time</th>
</tr>
</thead>
<tbody>
<tr>
<td>☐</td>
<td>262143</td>
<td>&gt; 1.5h</td>
</tr>
<tr>
<td>✓</td>
<td>83</td>
<td>9.75s</td>
</tr>
</tbody>
</table>

▶ Compression ratio (amount of nodes) of 99.97%
Conclusion

- Equivalent transformation of FF-DNNs into significantly and losslessly compressed DTs for better explainability
- Transformation algorithm and actual implementation for standard PyTorch models as input
- Evaluated for small model
- Observed very high compression ratio
- Seems to be a good trade-off between
  - Powerful, efficient-learnable DRL models and
  - Explainability of learned strategies
Bibliography I


Backup slides

Upon here, there are backup slides.
Linear transformation

\[ Y_1 = W_1^T X + B \quad Y_r = X W_r^T + B \]

\[
\hat{W}_i^T = \sigma(x_{i-1} W_{i-1}^T + B_{i-1}) W_i^T + B_i \\
= \sigma((W_{i-1} x_{i-1}^T + B_{i-1}^T)^T) W_i^T + B_i \\
= (a_{i-1} \odot (W_{i-1} x_{i-1}^T + B_{i-1}^T)) W_i^T + B_i \\
= (((a_{i-1} \odot (W_{i-1} x_{i-1}^T + B_{i-1}^T)) W_i^T + B_i \\
= (W_i (a_{i-1} \odot (W_{i-1} x_{i-1}^T + B_{i-1}^T)))^T + B_i \\
= (((W_i \odot a_{i-1})(W_{i-1} x_{i-1}^T + B_{i-1}^T))^T + B_i \\
= (((W_i \odot a_{i-1})(W_{i-1} x_{i-1}^T + B_{i-1}^T) + B_i^T))^T \\
= \left(\left(\left(W_1 \odot a_0\right) W_0 x_0^T + B_0^T + B_1^T\right) \ldots \right)^T \\
= \left(\left(W_1 \odot a_0\right) W_0 x_0^T + \left(W_1 \odot a_0\right) B_0^T + B_1^T\right) \ldots \right)^T \\

(1)

(2)
ARL Architecture

Figure. ARL Architecture [6]
Computation time of simple example DT with NN2EQCDT

Figure. Boxplot ($n = 30$) for the computation time of the NN2EQCDT algorithm for the simple model