NN2EQCDT

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

Torben Logemann [1] Eric MSP Veith [1]

[1] Carl 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 Reinforcment 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

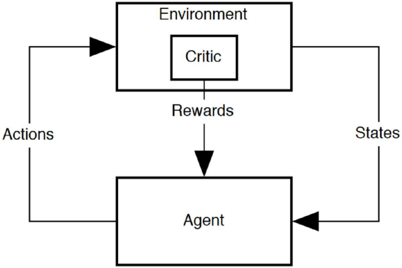


Figure. RL Architecture [1]

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

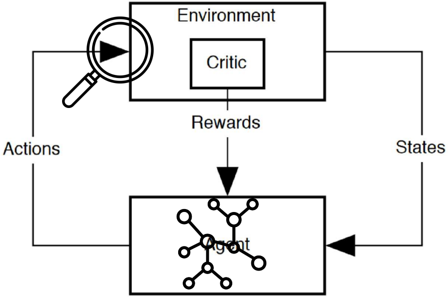
RL application fields

- 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

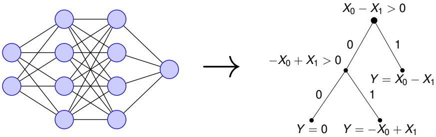
- Adversarial Resilience Learning (ARL) attack agents are deployed with the goal of causing voltage band violations [5]
- Explanation extracted by analysis of the impact of attacker actions on victim buses.
- Not deeply interpreted and no guarantees for all situations
- Guarantees important for defender 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 NN2EQCDT

```
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: $hid = 8$

NN2EQCDT algorithm

$\hat{\boldsymbol{W}} = \boldsymbol{W}_0$
$\hat{\boldsymbol{B}} = \boldsymbol{B}_0^\top$
$rules$ = calc_rule_terms($\hat{\boldsymbol{W}}, \hat{\boldsymbol{B}}$)
T, new_SAT_leaves = create_initial_subtree($rules$)
set_hat_on_SAT_nodes($T, new\_SAT\_leaves, \hat{\boldsymbol{W}}, \hat{\boldsymbol{B}}$)
for $i = 1, \ldots, n-1$ **do**
    SAT_paths = get_SAT_paths(T)
    for SAT_path in SAT_paths **do**
        $\boldsymbol{a}$ = compute_a_along(SAT_path)
        SAT_leave = SAT_path[−1]
        $\hat{\boldsymbol{W}}, \hat{\boldsymbol{B}}$ = get_last_hat_of_leave(T, SAT_leave)
        $\hat{\boldsymbol{W}} = (\boldsymbol{W}_i \odot [(\boldsymbol{a}^\top)_{\times k}])\hat{\boldsymbol{W}}$
        $\hat{\boldsymbol{B}} = (\boldsymbol{W}_i \odot [(\boldsymbol{a}^\top)_{\times k}])\hat{\boldsymbol{B}} + \boldsymbol{B}_i^\top$
        $rules$ = calc_rule_terms($\hat{\boldsymbol{W}}, \hat{\boldsymbol{B}}$)
        new_SAT_leaves = add_subtree($T, SAT\_leave, rules, invariants$)
        set_hat_on_SAT_nodes($T, new\_SAT\_leaves, \hat{\boldsymbol{W}}, \hat{\boldsymbol{B}}$)
convert_final_rule_to_expr(T)
compress_tree(T)

Figure 1. NN2EQCDT algorithm

Effective weight matrix calculation

$\hat{\boldsymbol{W}} = \boldsymbol{W}_0$
$\hat{\boldsymbol{B}} = \boldsymbol{B}_0^\top$
for $i = 0, \ldots, n-2$ **do**
    $\boldsymbol{a} = [\ ]$
    for $j = 0, \ldots, m_i - 1$ **do**
        if $(\hat{\boldsymbol{W}}_j \boldsymbol{x}_0^\top + \boldsymbol{B}_j^\top)^\top > 0$ **then**
            $\boldsymbol{a}.\mathrm{append}(1)$
        else
            $\boldsymbol{a}.\mathrm{append}(0)$
    $\boldsymbol{W}_{i+1} \in \mathbb{R}^{m_i \times k}, \boldsymbol{a} \in \mathbb{Z}_2^{m_i}$
    $\hat{\boldsymbol{W}} = (\boldsymbol{W}_{i+1} \odot [(\boldsymbol{a}^\top)_{\times k}])\hat{\boldsymbol{W}}$
    $\hat{\boldsymbol{B}} = (\boldsymbol{W}_{i+1} \odot [(\boldsymbol{a}^\top)_{\times k}])\hat{\boldsymbol{B}} + \boldsymbol{B}_{i+1}^\top$
return $(\hat{\boldsymbol{W}} \boldsymbol{x}_0^\top + \hat{\boldsymbol{B}})^\top$

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

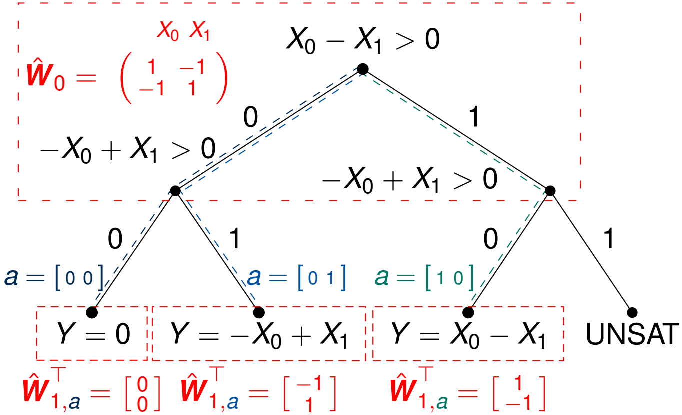


Figure. Simple example of an DT representing an XOR function constructed

XOR model: DT Compression

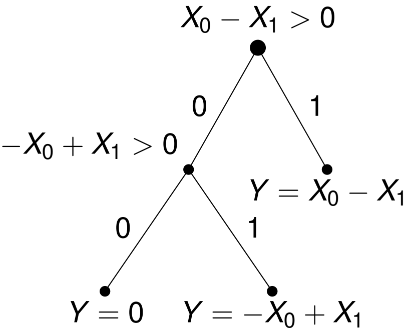
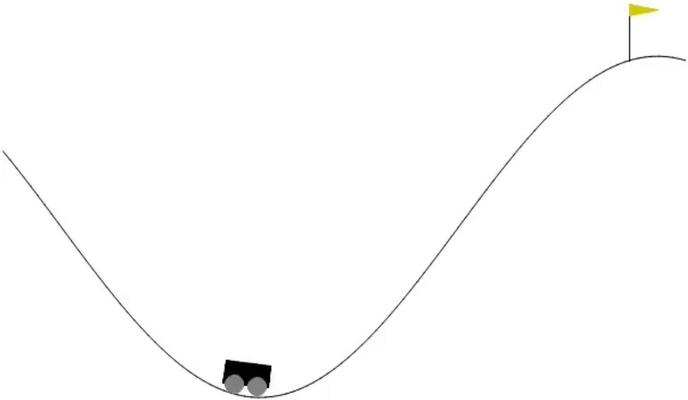
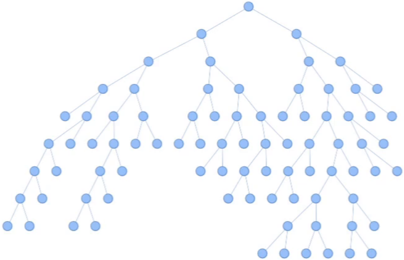


Figure. Simple compression example

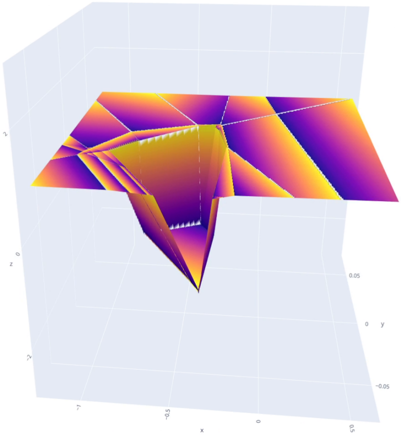
Simple example: Model car in MCC



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

Compression	$\#_{\text{nodes}}$	Computation time
☐	262143	$>$ 1.5h
☑	83	9.75s

- 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

[1] E. Puiutta and E. M. S. P. Veith, “Explainable reinforcement learning: A survey,” in *Machine Learning and Knowledge Extraction. CD-MAKE 2020*, vol. 12279, Dublin, Ireland: Springer, Cham, 2020, pp. 77–95. DOI: 10.1007/978-3-030-57321-8_5.

[2] M. Naeem, S. T. H. Rizvi, and A. Coronato, “A gentle introduction to reinforcement learning and its application in different fields,” *IEEE access*, vol. 8, pp. 209 320–209 344, 2020.

[3] A. E. Sallab, M. Abdou, E. Perot, and S. Yogamani, “Deep reinforcement learning framework for autonomous driving,” *Electronic Imaging*, vol. 29, no. 19, pp. 70–76, Jan. 2017, [retrieved: 05, 2023]. DOI: 10.2352/issn.2470-1173.2017.19.avm-023. [Online]. Available: https://library.imaging.org/ei/articles/29/19/art00012.

Bibliography II

[4] T. T. Nguyen and V. J. Reddi, "Deep reinforcement learning for cyber security," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–17, 2021. DOI: 10.1109/TNNLS.2021.3121870.

[5] E. M. S. P. Veith, A. Wellßow, and M. Uslar, "Learning new attack vectors from misuse cases with deep reinforcement learning," *Frontiers in Energy Research*, vol. 11, pp. 01–23, 2023, [retrieved: 05, 2023], ISSN: 2296-598X. DOI: 10.3389/fenrg.2023.1138446. [Online]. Available: https://www.frontiersin.org/articles/10.3389/fenrg.2023.1138446.

Bibliography III

[6] E. M. Veith, "An architecture for reliable learning agents in power grids," *ENERGY 2023 : The Thirteenth International Conference on Smart Grids, Green Communications and IT Energy-aware Technologies*, pp. 13–16, 2023, [retrieved: 05, 2023], ISSN: 2308-412X. [Online]. Available: https://www.thinkmind.org/articles/energy_2023_1_30_30028.pdf.

[7] J. Ba and R. Caruana, "Do deep nets really need to be deep?" *Advances in Neural Information Processing Systems*, vol. 27, pp. 2654–2662, 2014.

[8] Ç. Aytekin, "Neural networks are decision trees," *CoRR*, vol. abs/2210.05189, pp. 1–8, 2022, [retrieved: 05, 2023]. arXiv: 2210.05189. [Online]. Available: https://arxiv.org/abs/2210.05189.

Backup slides

Upon here, there are backup slides.

Linear transformation

$$\boldsymbol{Y}_\mathrm{l} = \boldsymbol{W}_\mathrm{l}^\top \boldsymbol{X} + \boldsymbol{B} \qquad \boldsymbol{Y}_\mathrm{r} = \boldsymbol{X}\boldsymbol{W}_\mathrm{r}^\top + \boldsymbol{B}$$

$$\begin{aligned}
\hat{\boldsymbol{W}}_i^\top &= \sigma(\boldsymbol{x}_{i-1}\boldsymbol{W}_{i-1}^\top + \boldsymbol{B}_{i-1})\boldsymbol{W}_i^\top + \boldsymbol{B}_i \\
&= \sigma((\boldsymbol{W}_{i-1}\boldsymbol{x}_{i-1}^\top + \boldsymbol{B}_{i-1}^\top)^\top)\boldsymbol{W}_i^\top + \boldsymbol{B}_i \\
&= (\boldsymbol{a}_{i-1} \odot (\boldsymbol{W}_{i-1}\boldsymbol{x}_{i-1}^\top + \boldsymbol{B}_{i-1}^\top)^\top)\boldsymbol{W}_i^\top + \boldsymbol{B}_i \\
&= ((\boldsymbol{a}_{i-1}^\top \odot (\boldsymbol{W}_{i-1}\boldsymbol{x}_{i-1}^\top + \boldsymbol{B}_{i-1}^\top))^\top)\boldsymbol{W}_i^\top + \boldsymbol{B}_i \\
&= (\boldsymbol{W}_i(\boldsymbol{a}_{i-1}^\top \odot (\boldsymbol{W}_{i-1}\boldsymbol{x}_{i-1}^\top + \boldsymbol{B}_{i-1}^\top)))^\top + \boldsymbol{B}_i \\
&= ((\boldsymbol{W}_i^\top \odot \boldsymbol{a}_{i-1}^\top)^\top(\boldsymbol{W}_{i-1}\boldsymbol{x}_{i-1}^\top + \boldsymbol{B}_{i-1}^\top))^\top + \boldsymbol{B}_i \\
&= ((\boldsymbol{W}_i \odot \boldsymbol{a}_{i-1})(\boldsymbol{W}_{i-1}\boldsymbol{x}_{i-1}^\top + \boldsymbol{B}_{i-1}^\top))^\top + \boldsymbol{B}_i \\
&= (((\boldsymbol{W}_i \odot \boldsymbol{a}_{i-1})(\boldsymbol{W}_{i-1}\boldsymbol{x}_{i-1}^\top + \boldsymbol{B}_{i-1}^\top)) + \boldsymbol{B}_i^\top)^\top \qquad (1)
\end{aligned}$$

$$\begin{aligned}
\mathrm{NN}(\boldsymbol{x}_0) &= (\dots((\boldsymbol{W}_1 \odot \boldsymbol{a}_0)(\boldsymbol{W}_0\boldsymbol{x}_0^\top + \boldsymbol{B}_0^\top) + \boldsymbol{B}_1^\top)\dots)^\top \\
&= (\dots(\underbrace{(\boldsymbol{W}_1 \odot \boldsymbol{a}_0)\boldsymbol{W}_0}_{\hat{\boldsymbol{W}}_{1,a_0}}\boldsymbol{x}_0^\top + \underbrace{(\boldsymbol{W}_1 \odot \boldsymbol{a}_0)\boldsymbol{B}_0^\top + \boldsymbol{B}_1^\top}_{\hat{\boldsymbol{B}}_{1,a_0}})\dots)^\top
\end{aligned} \qquad (2)$$

ARL Architecture

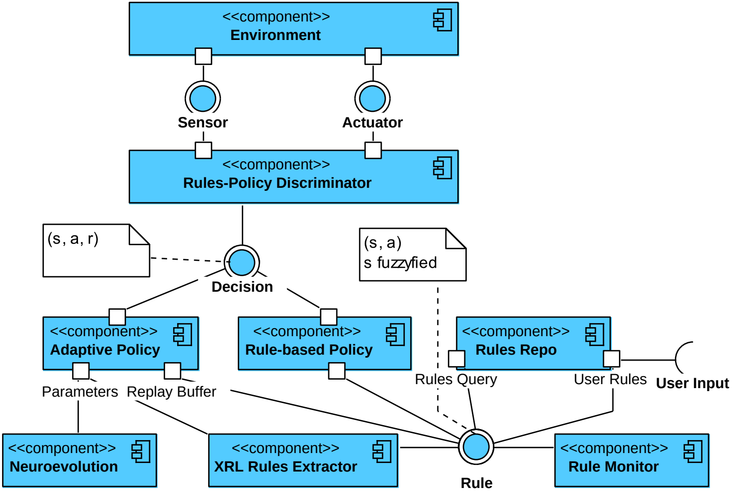


Figure. ARL Architecture [6]

Computation time of simple example DT with NN2EQCDT

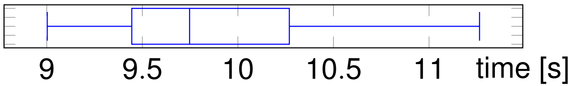


Figure. Boxplot ($n = 30$) for the computation time of the NN2EQCDT algorithm for the simple model