Explainability First! Cousteauing the Depths of Neural Networks to Argue Safety

Markus Borg[1]

“Safety first!” is a weary expression that has been repeated until tedium. But any organization trying to obtain a safety certification for a software-intensive system knows that safety is achieved first *after* considerable effort. Hard evidence is required for safety engineers to develop a safety case, i.e., a structured argumentation for why a system is adequately safe in a specific context. Thus, in the case of Machine Learning (ML)-based Cyber-Physical Systems (CPS), a better expression would be “Explainability first!” — at least in the eyes of external safety assessors.

ML-based systems will be increasingly used in safety-critical applications, e.g., in Automated Driving (AD). Increasing the level of automation has great potential to improve safety by mitigating risks introduced by humans. Human factors such as tiredness, distractions caused by smart phone usage, vivid conversations, and loud music do not affect automated vehicles. In Sweden, the most alarming studies estimate that the number of cars driven by influenced drivers matches the number of taxis in operation[2]. While there are obvious advantages with AD, how could you tackle the challenge of developing the required safety argumentation?

#problem We are currently studying Deep Neural Networks (DNN) trained to enable vehicle environmental perception, i.e., awareness of elements in the surrounding traffic, and how to approach the corresponding safety certification. Successful perception is a prerequisite for AD features such as lane departure detection, path/trajectory planning, vehicle tracking, and scene understanding. A wide range of sensors have been used to collect input data from the environment, but the most common approach is to rely on front-facing cameras. **#deeplearning #automotive #vision #safetycertification**

DNNs have been reported to deliver superhuman classification accuracy for specific tasks, but inevitably they will occasionally fail to generalize. Unfortunately, from a safety perspective, analyzing when this might happen is currently not possible due to the black-box nature of DNNs [Bo]. A state-of-the-art DNN might be composed of hundreds of millions of parameter weights, thus the approaches to verification and validation of DNN components must be different compared to approaches for human-readable source code. Techniques

[1] RISE Research Institutes of Sweden AB, Software and Systems Engineering Lab, Scheelevägen 17, SE-223 70 Lund, Sweden markus.borg@ri.se

[2] https://www.dt.se/artikel/dalarna/15-000-resor-per-dygn-med-alkoholpaverkade-forare

enforced by the current safety standard for automotive systems, ISO 26262, are not directly applicable to DNNs, e.g., source code reviews and exhaustive coverage testing.

#expertise Apart from machine learning, we have experience in software engineering and safety-critical development. Functional safety is defined in ISO 26262 as "absence of unreasonable risk due to hazards caused by malfunctioning behavior of electrical/electronic systems". The fundamental assumptions are that zero risk can never be reached, but non-tolerable risks must be reduced "as low as reasonably practicable". The basic approach to achieve this is by developing an understanding of situations that lead to safety-related failures and, subsequently, designing the software so that such failures do not occur. Even if the system does not conform to its specification, it should not hurt anyone or anything. **#machinelearning #requirements #testing #traceability**

Hazards, i.e., system states that might lead to an accident, play a key role in safety-critical development. In the automotive domain, hazards are typically analyzed in four steps of early requirements engineering: 1) Hazard identification to populate a hazard register, 2) Hazard assessment of probability, acceptability, and controllability, 3) Hazard analysis to identify root causes, and 4) Risk reduction by specifying safety requirements that should ensure that the system remains safe even when things go wrong. Complete traceability from safety requirements to their software (or hardware) implementation and their corresponding test cases constitute the backbone evidence of the safety case, i.e., the documented argumentation that a system is safe, provided to external safety assessors. But what happens when a trace link from a safety requirement ends in a DNN? [BED17]

#need We need help to develop safety cases that encompass ML, i.e., complex models that are trained on massive annotated datasets. Analogous to how the French oceanographer Jacques Cousteau pioneered charting the depths of the great oceans, the software engineering community needs to dive into the state-of-the-art DNNs to explain their intricacies. We must understand how deeply we should trace from critical requirements to explain and argue safety of an ML-based system. Should we trace to source code calling an ML library? The underlying ML model? The model architecture? Its parameter weights? Or even to specific examples in the training data? We might need traceability to several of the listed examples, since safety evidence tends to aim at exhaustiveness. Surely, before any safety certification of ML-based systems can be obtained, we first need to find ways to explain why they are safe. **#safetycase #certification #standards #evidence**

References

[BED17] Borg, M.; Englund, C.; Duran, B.: Traceability and Deep Learning - Safety-critical Systems with Traces Ending in Deep Neural Networks. In: Proc. of the Grand Challenges of Traceability: The Next Ten Years. pp. 48–49, 2017.

[Bo] Borg, M. *et al.*: Safely Entering the Deep: A Review of Verification and Validation for Machine Learning and a Challenge Elicitation in the Automotive Industry. Journal of Automotive Software Engineering, Under revision.