Requirements Assurance in Machine Learning

Alec Banks and Rob Ashmore

Defence Science and Technology Laboratory, Salisbury, United Kingdom
abanks@dstl.gov.uk, rdashmore@dstl.gov.uk

Abstract

Training data is an important aspect of approaches that use Machine Learning techniques. More precisely, we assert that training data captures the requirements that should be satisfied by the trained algorithm. Hence, for safety applications, any argument relating to behavioural correctness has to consider how those requirements are embodied within the training data. To support this, based on approaches for requirements assurance in traditional safety-related software, we develop nine specific areas where confidence is required in training data. These are illustrated using a fictional example.

Introduction

This paper outlines the challenges associated with the assurance of requirements in safety-related Machine Learning (ML) systems.

All software operates within the context of the system in which it is executed. In traditional safety-related systems the behavioural requirements are first established at the system level and then decomposed and refined until such a time where the developer is able to unambiguously transfer the associated requirement into code. In ML-based systems the software behaviour is not dictated by requirements that have been decomposed to that level (Ashmore and Lennon, 2017). Instead, those requirements are implicitly provided via the training data.

This paper briefly looks at the way requirements are treated in existing safety standards. It goes on to discuss the concept of requirements in developments based on ML techniques. Building from existing approaches, a series of assurance considerations for ML requirements are developed and illustrated using a fictional, indicative example.

Treatment of Requirements in Established Software Standards

There are a number of software standards that may be applied to safety-related applications. These all seek to avoid the introduction of errors and foster their rigorous removal. Whilst they are often domain specific, all of these standards have common characteristics, that were distilled into the '4+1' principles of software safety engineering (Hawkins *et. al*, 2013). To achieve all of these principles, software safety assurance must:

- P1. Identify safety requirements at the system level;
- P2. Maintain the intent of these requirements throughout decomposition;
- P3. Demonstrably satisfy safety-related requirements in the implementation;
- P4. Identify hazardous behaviours introduced by the software and mitigate them; and
- P4+1. Provide a level of confidence in software behaviour that is commensurate with its contribution to system-level risk.

In safety-related applications these principles usually drive the software requirement decomposition to two distinct levels. High-Level Requirements (HLR) detail 'what is required' in the design. These are then systematically decomposed into Low-Level Requirements (LLR), which provide coders with information on 'how to implement' that design. To minimize ambiguity LLR often include pseudo-code or mathematical formulae.

Requirements Definition in an ML Context

In ML applications, the requirements for the software can be considered from two parallel, but related, perspectives. There are, firstly, the requirements for the construction of the learning algorithm and, secondly, the general requirements for behaviour. We use the term 'general' purposefully here. If it is possible to detail the exact behaviours expected from the software then ML approaches are arguably inappropriate for safety-related applications (Salay and Czarnecki, 2018).

For construction of the learning algorithm (e.g. back-propagation) it is possible (and desirable) to develop HLR and to further decompose these into LLR and onto implementation, verification and validation. This aspect of ML software is therefore not considered further in this paper, although we note that work is being conducted in this area, for example (Srisakaokul *et. al*, 2018).

Moving on to the more challenging area of the behaviour of the trained algorithm, in an ML-based approach this is dictated by the training data, which may be real and/or synthetically generated (e.g. Ekbatani, *et. al*, 2017, for coputer vision problems), combined with the learning algorithm and the structure (e.g. number of neurons and layers in an artificial neural network) to which it is applied.

From this short discussion and returning to the '4+1' principles, it is apparent that in the case of ML, principles 1 and 4+1 are arguably the only ones that can, based on current practices, be adequately satisfied. Principles P2, P3 and P4 all suffer to a greater or lesser degree because: the exact behaviour cannot be detailed in LLR (P2); assured through verification (P3); or sufficiently predicted to permit potential hazard identification (P4). We note that some of these may be resolved through behavioural containment (e.g. the use of monitors and alternative control structures within the wider system architecture) but to do so could also negate the benefits of adopting ML in the design.

Due to the challenges of addressing each of these principles individually, we suggest a holistic perspective may be more beneficial. Whilst all of the principles relate to assurance of requirements and their implementation, discussing all of them would be a significant endeavor, which would be too broad for this position paper. Consequently, we focus on the decomposition of the HLR into LLR (i.e. assurance of the requirements rather than their initial derivation or final implementation).

Since it determines the algorithm's behaviour, we can consider the training data to be an abstract form of the LLR. Hence, assurance of training data is paramount to gaining confidence. Using RTCA DO-178C (RTCA, 2011) as an example[1], traditional requirements verification seeks to ensure LLR are:

- R1. Compliant with HLR;
- R2. Accurate and consistent;
- R3. Compatible with target computer[2];
- R4. Verifiable;
- R5. Conforming to standards;
- R6. Traceable; and
- R7. Algorithmically correct.

Whilst all of these considerations arguably apply to ML-based applications some do not easily translate. To achieve the same intent through the training data we need to have confidence that the data:

- D1. Relates to the intent of the HLR;
- D2. Does not contain bias;
- D3. Is sufficient;
- D4. Is syntactically and semantically correct;
- D5. Addresses normal and robustness behaviours;
- D6. Is self-consistent;
- D7. Conforms to standards;
- D8. Is compatible with target computer; and
- D9. Is verifiable.

For ease of reference, Table 1 summarises the relationship between requirements traditionally placed on LLR (R1 to R7) and areas where confidence is needed in the training data (D1 to D9).

Table 1: Relationship between traditional requirements and areas of confidence in training data

	R1	R2	R3	R4	R5	R6	R7
D1						X	X
D2							X
D3	X						
D4		X					X
D5	X						
D6		X					
D7					X		
D8			X				
D9				X			

It is readily apparent that each of the traditional LLR requirements is covered by at least one area of training data confidence. This suggests the identified areas are necessary. It does not mean they are sufficient (or complete).

For example, there are also several 'meta-considerations' (i.e. those not directly concerned with performance but contribute to design confidence). These include, *inter alia*: the data source; its control; forensic auditability; extent of pre-processing required; *etc*.

To provide confidence that these areas are suitably addressed, there are a number of processes, reviews, analyses and tests that can be applied, the results of which should support a safety argument. Some of these approaches will now be discussed in the context of an indicative example.

Indicative Example

To colour our discussion we adopt a fictional landing system that detects whether a medium size Unmanned Air

[1] DO-178C is a key software safety document for aircraft. It is sufficiently general to apply to most developments of safety-related software.
[2] The target computer is the one on which the algorithm will run during operational use. This is often different from the host computer, which is the one used to develop the algorithm.

Vehicle (UAV) has landed on an unprepared surface. The associated system requirement might be:

- SYS-01: Detect landing on unprepared strip.

This would then be decomposed into software and hardware requirements. Given the environmental constraint of the unprepared strip the conventional 'Weight-On-Wheels' switch[3] approach would not be technically feasible. Therefore a design decision is made to use a combination of hardware sensor systems (e.g. horizontal and vertical accelerometers, altimeters, air data *etc*.) as inputs to an Artificial Neural Network (ANN) that will classify the landing status into one of the following categories: in-air; on approach; landing; landed.

The hardware requirements are not considered further, but the high-level software requirement (HLR) might be:

- SW-HLR-01: Classify landing status: {*In_Air*; *On_Approach*; *Landing*; *Landed*} based on data from: Inertial Navigation System {*Vert_Acc*; *Horz_Acc*}; Laser Altimeter {*Height*}; Air Data Computer {*Airspeed*}.

The actual behaviour (LLR) of the ANN would be determined by the training data (and the learning algorithm and the structure to which it is applied). For the purposes of this discussion we assume data has been collected from a suitable light aircraft making a number of landings on a variety of surfaces in a range of environmental conditions. For synthetic data, it too must be verified and validated to ensure that its form (syntax) and intent apropos HLR (semantics) matches the expectation of real data. Approaches to this are not expected to differ from those discussed herein to ensure data correctness.

In our example, the number of operationally-generated samples is likely to be limited, partly through cost and partly through practicality (e.g. it is unlikely that there would be many, if any, landings performed with systems deliberately set in failure modes, even if this were possible). Consequently, the recorded (operationally-generated) data is applied to a generative data modelling tool that provides a larger data set which is used to train the ANN.

To determine whether the nine areas of training data confidence have been addressed a series of reviews, analyses and tests are required. The following paragraphs discuss the most significant of these; space limitations prevent a complete analysis of all nine areas.

For operationally-generated samples, traceability in the intent of data (D1) is straightforward, although unusual, outlier examples may need close examination to assure validity. For synthetically generated data, ensuring the intent of the data would rely on detailed knowledge of the behaviour of both the UAV and the data-generating tool. Additionally, most safety-related systems also feature robustness requirements, which would need to be implemented through the inclusion of sufficient examples in the data (robustness is also considered in the discussion of D5, below).

The diversity of potential environmental conditions compared to those that might be available during training data collection would be a prime breeding ground for unintentional bias (D2). When this limited dataset is then introduced into the generative model the potential becomes amplified. Detecting bias in data is difficult, but there are a number of ways it could be detected in the trained algorithm (Tan, 2017).

Whilst clearly there is a numerical aspect to the sufficiency of training data (D3), we are also concerned with the diversity of input data. Identifying areas of sparsity is a notable concern. In our indicative example, there may be limitations on the environmental conditions in which data can be collected. Understanding this may lead to restrictions being placed on operational use of the UAV until further data can be collected.

If a formally-structured process is used to record operational data and strong configuration control is applied then data taken from real systems should be syntactically and semantically correct (D4). However, since our example includes synthetic data, care needs to be applied to ensure that data distributions match real world expectations. Syntactic aspects relate to the structure and ranges of data, which can be tested using simple data analysis tools. The semantic aspects are more challenging and should include tests for data poisoning[4] and unintentional examples where small shifts in inputs cause large changes in output. Guidance for the management of safety-related data has been produced by several organisations, e.g. (DSIWG, 2018), and can be looked to for assistance in minimizing the potential for data poisoning. However, the developing body of knowledge in this area indicates that a stronger argument may be possible through the application of data poisoning detection tools (Steinhardt *et. al*, 2017).

A safety-related system that only contains requirements to address normal range behaviour has the potential to become unsafe under abnormal conditions. Traditional system requirements software design should consider all reasonable failure conditions. In our indicative example robustness cases (D5) would include sensors providing inaccurate readings as well as total failures (which may mean no reading is available). Inaccurate readings include credible but incorrect, as well as incredible data. As with traditional designs, a conscious decision needs to be made about the extent to which the system can be expected to deal with failure conditions. Our indicative landing system takes four inputs; it might be reasonable to expect that one input could fail at any given time so data that represents

[3] Usually consisting of a proximity switch located on the undercarriage, which uses movement induced when weight is applied to the wheel to make an electrical circuit.

[4] Where an attacker can alter a small fraction of the training data.

landings where *Height* = 100ft but all other sensors are indicative of a landing would be a reasonable robustness case. However, simultaneous failure of three inputs would be unreasonable. Synthetic data is expected to be invaluable in generating sufficient data for training systems to behave robustly, since real data may be dangerous to collect.

Self-consistency (D6) can be more challenging than it might first appear. The large number of landings in multiple environments means that some results might be seemingly contradictory; equivalently, based on the available data the landing classes may not be separable. For example, a really smooth landing in wet conditions may lead to input signals similar to *In-Air* (e.g. laser altimeter might read high altitude due to spurious reflections and accelerometers read low *Vert_Acc* and *Horz_Acc*) but with a label of *Landing*. If that is the case, the underlying features of the contradictions may need to be explored and the system may need to be redesigned (e.g. to provide additional sensor inputs to introduce new variables) for greater robustness. Fortunately, statistical analysis of the data may help. Data that is distributionally dissimilar but has the same classification may be inconsistent. In many ways this can be seen as the inverse problem to detecting adversarial examples, where the distribution is similar but the classification is erroneously different.

The final three considerations (D7, D8 and D9) are regarded as unchanged from traditional safety-related software and are not discussed further. However, we note that the concept of verifiability is potentially different for ML-based systems in that training data seeks to implement general behaviours and direct verification may not be possible.

Summary and Conclusion

This short position paper has established the concept that training data provides the functional requirements for a safety-related system developed using ML-based approaches.

It has shown that it may be possible to make claims that the intent of HLR, passed down from system level requirements, have been correctly maintained and implemented through the training data. Using traditional assurance concepts as a basis we have developed a series of training data considerations that we argue could form the basis of an assurance activity.

These considerations can be addressed by a combination of sound data management and a collection of reviews, tests and analyses. Some of these are currently under development but further work is required to develop a comprehensive toolset that may be used across a wide range of data sets.

In conclusion, we assert that any assurance claims regarding the requirements aspects of an ML-based safety-related system would, as a bare minimum, need to address the nine areas developed in this paper. We also encourage the safety-related ML community to test these areas, developing them further as required.

References

Ashmore, R. and Lennon, E. 2017. Progress Towards the Assurance of Non-Traditional Software. In *Developments in System Safety Engineering, Proceedings of the Twenty-fifth Safety-Critical Systems Symposium*. Bristol, UK.

Certification Authorities Software Team (CAST). 2003. Merging High-Level and Low-Level Requirements. *Position Paper CAST-15*, completed February 2003.

Data Safety Initiative Working Group (DSIWG). 2018. Data Safety Guidance, SCSC-127B, ISBN 978-1540887481.

Ekbatani, H.K., Pujol, O. and Segui, S. 2017. Synthetic Data Generation for Deep Learning in Counting Pedestrians. In *6th International Conference on Pattern Recognition Applications and Methods. SCITEPRESS - Science and Technology Publications.*

Hawkins, R., Habli, I., and Kelly, T. 2013. The Principles of Software Safety Assurance. *31st International System Safety Conference.* Boston, Massachusetts USA, 2013.

RTCA. 2011. Software Considerations in Airborne Systems and Equipment Certification. DO-178C.

Salay, R., and Czarnecki, K. 2018. Using Machine Learning Safely in Automotive Software: An Assessment and Adaption of Software Process Requirements in ISO 26262. *arXiv preprint arXiv:1808.01614.*

Srisakaokul, S., Wu, Z., Astorga, A., Alebiosu, O., and Xie, T. 2018. Multiple-Implementation Testing of Supervised Learning Software. In *Proc. AAAI-18 Workshop on Engineering Dependable and Secure Machine Learning Systems (EDSMLS)*.

Steinhardt, J., Koh, P. W. W., and Liang, P. S. 2017. Certified Defenses for Data Poisoning Attacks. In *Advances in Neural Information Processing Systems* (pp. 3517-3529).

Tan, S., Caruana, R., Hooker, G., and Lou, Y. 2017. Detecting Bias in Black-Box Models using Transparent Model Distillation. *arXiv preprint arXiv:1710.06169*.

Disclaimer