Quality Assurance of Machine Learned Models by Integrating Domain Knowledge and Formal Verification

Rüdiger Ehlers[1] Jörg Grieser Christoph Knieke Andreas Rausch Mirco Schindler[2]

A large number of software systems nowadays employ artificial neural networks and other types of learned models from the area of machine learning. Such models are inferred from the available data and then used for classification and forecasting purposes.

While modern machine learning algorithms are highly scalable and able to deal with huge amounts of input data, they do not guarantee that the learned models capture the core aspects of the processes from which the data used for learning was acquired. Statistical approaches such as measuring the accuracy of a learned model on test data provide some insight into whether the model generalizes well, but the data set from which the model is learned is typically quite limited in size, which can impact its generalization capabilities and induces the need for quality assurance of learned models. Unlike in traditional software engineering, where code reviews and model-based testing are available as suitable quality assurance methods, learned models lack a structure that supports their manual review, and comprehensive specifications for exhaustive testing are seldom available.

So given that quality assurance is necessary, how can we interpret and explain the behavior of learned models? We argue in this talk that (1) the integration of *domain knowledge* into the learning process combined with (2) the employment of *formal verification* approaches to derive useful information about the learned model allows to explain the *behavior* of a model (as opposed to the model itself) to a large degree.

Let us discuss this idea by means of an example. Assume that a bank wants to predict whether a customer will default on a loan or not. There is a history of cases available in which for every customer, the number of years already working at the current employer is known in addition to the income per year. Additionally, it is known whether the customer defaulted, had only some delays paying the installments, or always paid the installments on time.

We can use a standard neural network architecture and a standard learning algorithm to compute a classifier from such a data set that maximizes the number of historical cases in which it is correctly predicted whether the customer defaulted or not (i.e., the credit worthiness). By doing so, we however ignore the possibility to integrate domain knowledge,

[1] Universität Bremen, Fachbereich für Mathematik & Informatik, Bibliotheksstraße 5, 28359 Bremen, ruediger.ehlers@uni-bremen.de
[2] Technische Universität Clausthal, Institut für Software and Systems Engineering, Arnold-Sommerfeld-Straße 1, 38678 Clausthal-Zellerfeld, Germany, <firstname>.<lastname>@tu-clausthal.de

doi:10.18420/se2019-49

namely that a higher yearly income should never decrease the credit worthiness. Hence, the classifier should be *monotone* in one of the input dimensions. For the number of years spent at the same employer, we may not have any insight in monotonicity, e.g., a very high number of years spent could decrease the probability of the employee finding work elsewhere in case his/her employer bankrupts. Hence, we only know that the learned model should be monotone in one dimension. Such a monotonicity requirement is relatively easy to integrate in modern *deep learning* toolkits, as we show in this talk.

Once we ensured that the learned model is partially monotone, it makes sense to ask questions such as "starting from which yearly income is a customer definitely classified in the best credit worthiness category when being with the same employer for the last 5-10 years" to gain insight into whether the learned classifier has reasonable behavior. While the question is also reasonable to ask without ensuring monotonicity, a monotone network is less likely to have outliers, i.e., small regions of the input space that are surrounded by a part of the input space that the network maps to a different classification. Such outliers may be small but make the answers to such queries less useful.

To obtain the answers to such questions on learned networks in applications that do not require frequent re-learning of the model, algorithms from the quickly growing area of *neural network verification* can be applied. They are currently limited to relatively small network sizes and hence have limited use for current computer vision applications. Yet, for many decision making applications, small networks often suffice and can already be analyzed by contemporary tools.

More challenging are systems that learn during operation. In this case, runtime monitoring is a promising approach. The runtime monitor verifies rules based on the domain knowledge, such as the existence of a pre-defined income threshold above which a customer is always seen as creditworthy. Due to the fact that the neural network can change its behavior almost arbitrarily, continuous verification is required. For a centralized system such as a loan determination system this is conceivable.

In case of restrictions such as limited computing resources, another approach can be followed. For instance, we can define a space of permitted network behavior such that verification is only triggered if an output leaves this space. In this way, we are able to verify behavioral properties and ensure that the network operates within reasonable limits.

We close the talk by giving an outlook on some additional challenges for quality assurance of systems involving learned models. For instance, in the currently running "*MaMMa : Maintained Mine and Machine*" project[3], we deal with the predictive maintenance problem for complex machines. Since maintenance is costly, the decision suggested by a learned model can only be followed if a reason is provided that is understandable by a machine engineer. As a consequence, we need to restrict the learning process to models that permit reason extraction.

[3] `https://mamma-project.eu`