University of St Andrews

Property Infererence for Deep Neural Networks: A Review

Saurav Jha

sj84@st-andrews.ac.uk

December 18, 2020

1 Introduction

Following their recent boom, Deep Neural Networks (DNNs) are being extensively used across industries ranging from education to health care and from weather to stock market predictions. All these models are universal approximators as they can approximate any function F which maps an n-dimensional input vector of real values R^n to an m-dimensional output vector of real values R^m, i.e. $F : R^n \longrightarrow R^m$. However, DNNs are largely black-boxes since the mere study of their structure do not throw light on the form of F given no simple link between the weights of the model and the nature of the function. This has partly led deep learning models struggle to find their place for safety-critical purposes. As DNNs work by iteratively adjusting their weights based on the input instead of asking *if-then* questions, their decisions are often inexplicable when presented with an input that is outside the distribution of its training data. This can be made clear by figure 1 where Brown et al. (2018) show how the placing of a sticker physically next to a banana convinces the classification model - that had never seen the image of a sticker during training - to predict it a toaster with 99% confidence.

With the rise of data as the most valuable resource of the world, past years have witnessed a growing awareness for data regulation among the scientific community and the governments. That said, neural networks requiring large volumes of data to be trained are at the forefront of the quest for explainable AI thus attracting several big-budget projects such as the EU's High-Level Expert Group on Artificial Intelligence[1]. While such *explainability* is a desired property for all DNN models, their extent might vary based on the applications, *i.e.* a pharma company must consider understanding the decisions of a DNN as it can affect lives while for churn prediction, interpretability can be secondary. For dependable systems leveraging DNNs, explainability is crucial in pointing out the unanticipated situations where they might fail and thus can count towards achieving the core principles of safety, reliability and security. An example depicting the importance of this is evident from test of Zhou and Sun (2019) for the real-life LiDAR obstacle perception system of Baidu's Apollo self-driving software. The work found 24 out of 1000 such test cases where even a single random point - as tiny as an air particle - scattered outside the driving area could cause an autonomous vehicle to fail to detect an obstacle on the roadway.[2] While this approach takes into consideration the design of test cases that accurately reflect the environmental conditions in which the DNN will operate,[7] such methods may be insufficient in encompassing every possibility given the combinatorial explosion in real-world data points. Consequently, the need for approaches inferencing the parameters of models is imminent.

[1]https://ec.europa.eu/digital-single-market/en/high-level-expert-group-artificial-intelligence

[2]More information at: https://cacm.acm.org/magazines/2019/8/238335-a-case-against-mission-critical-applications-of-ma
fulltext.

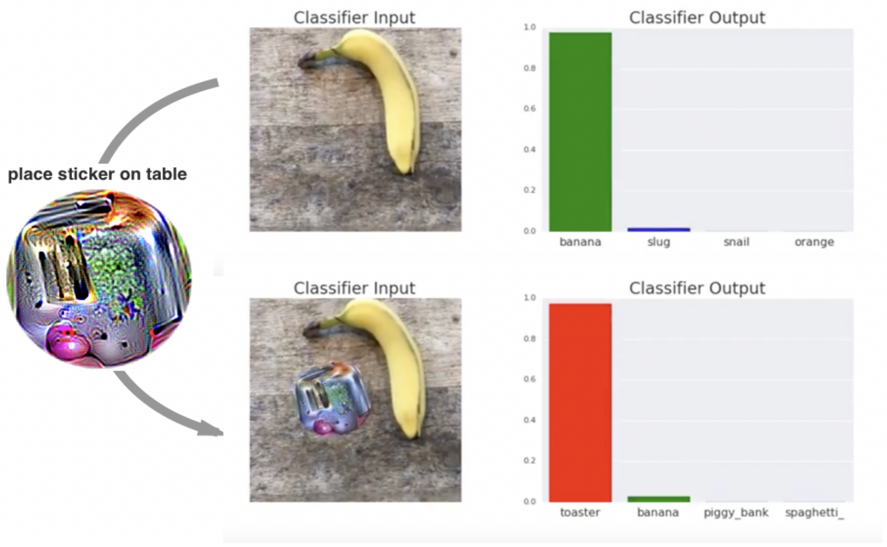


Figure 1. Figure showing a flaw in interpretation of VGG16 model output: adapted from Brown et al. (2018).

1.1 Formal methods for model inference

In the midst of several works targeting to solve the black-box property of DNNs, the wider field of software verification through formal methods such as *decision patterns*, which are often used to improve the quality of safety-critical software systems [5], remain untouched. Gopinath et al. (2019) describe a first step towards inferring the working of DNNs as formal properties in the form $Pre \implies Post$. Their work targets to infer the condition *Pre* as a formal explanation for the condition *Post* which refers to the desired behavior of the network -*i.e.*, if the predicted class is *'C'*. In doing so, the paper[3] treats a DNN as a feature extractor (i.e., a neuron takes input X, multiplies it by weight w, adds bias b and applies a non-linear activation function before propagating the result to a neuron in the next layer) and formulates *input* and *layer* properties based on these. While both of them imply the desired output behavior, *input properties* hold for the original input space while *layer properties* group inputs that result in common characteristics among features extracted at an intermediate layer. Using the example of a simple feed forward network with two inputs, outputs, and hidden layers each, and a special case of activation function - Rectified Linear Unit (ReLU), the work uses decision patterns of individual neurons (*i.e.,* values *on* and *off* indicating whether the neuron propagates its results to the next layer or not) as preconditions given that such patterns can group inputs that are processed by the same network of neurons and return the same output. The grouping then helps form convex predicates in the corresponding space which can be solved efficiently using linear programming solvers. As depicted in figure 2, input properties define convex regions in the original input space, viz. *X1* and *X2*, by constraining activation status of all neurons up to an intermediate layer, viz. layer 1, while layer properties do the same at an intermediate layer and thus can be expressed as unions of convex regions in the input space (e.g., the neuron $N_{1,1}$ in hidden layer 1 is turned on by both the inputs *X1* and *X2* and hence its convex region in the input space is marked by the union of green and blue lines spanning over X1 and X2).

The *decision patterns* of a DNN can thus be thought as its program paths influencing different behaviors of the network. The paper intends to explore input-output properties that can explain these behaviours by formally specifying them at the same time. To this goal, it proposes two such methods: (a) iterative refining of decision procedures using an example of Reluplex [6], and (b) learning a decision

[3]The term "paper" refers to the work of Gopinath et al. (2019) hereafter.

tree model from the data followed by either a formal or empirical validation of the learned patterns. The paper then applies these to learning input and output properties for classification on MNIST dataset as well as learning intermediate properties for a safety-critical system for an unmanned aircraft control, *viz.* ACAS XU.

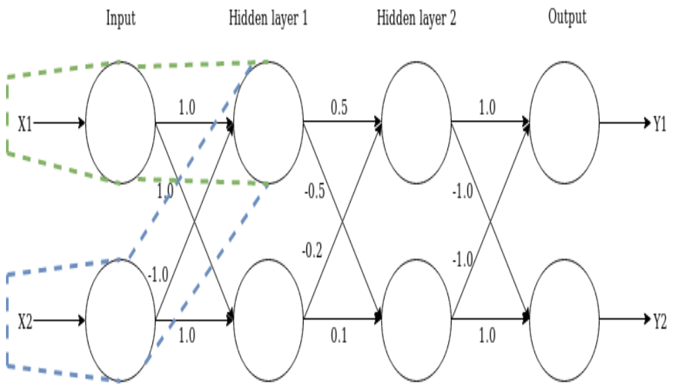


Figure 2. Figure showing a feed-forward DNN with convex blue and green regions for input [1.0, -1.0]: network configurations adapted from Gopinath et al. (2019).

1.2 Network Properties

Network properties group together inputs which are governed by the same decision pattern and hence, produce the same output. Since the output of applying ReLU function to an input X is either 0 (if X $\leq$ 0) or positive (if X $>$ 0), such a decision pattern σ for a network with ReLU activation comprise of subsets of neurons with activation statuses *{on, off}* -*i.e.*, neurons marked on are denoted by *on(σ)* while those off by *off(σ)*. Therefore, the predicate for decision pattern over the set of all inputs X is a logical conjunction of neurons N with zero or greater outputs:

$$\sigma(X) ::= \bigwedge_{N \in on(\sigma)} N(X) > 0 \;\; \wedge \bigwedge_{N \in off(\sigma)} N(X) = 0 \tag{1}$$

Further, a *decision pattern* is treated as a *network property* wrt to a postcondition P if it holds good for all inputs X, i.e. σ(X) $\Longrightarrow$ P(F(X)) for all X. Out of all such decision patterns, the work aims to study *minimal patterns*, *i.e.* pattern with no redundant neuron whose dropping would invalidate it, since these have no unnecessary constraints and ensure that more inputs satisfy the property, which the work refers to as the *support* of a pattern, *supp(σ)*. More formally, *supp(σ)* is the probability mass of inputs satisfying σ in that it measures the number of inputs out of total inputs for which σ holds true.

The paper puts forward two such propositions, each of which correspond to the two network properties - the input and the layer properties:

1. ***Proposition 1*** derives the relation for any input *X'* and a convex postcondition P given that P holds true for at least one other input X -*i.e.*, P(F(X)) should exist. Mathematically, if σ(X') $\longrightarrow$ F(X') = W.X' + b then the predicate σ(X') $\wedge$ P(W.X' + b') is an input property.

2. ***Proposition 2*** states that a layer property σ^l for an intermediate layer l can be a grouping of several input properties, i.e. σ^l(X) $\Longrightarrow \bigvee_i(\sigma^l(X) \wedge (\sigma_i(X))$. The depiction for this follows figure 2.

Explanation: Proposition 1 can be established from the example depicting the convex regions in the input space that contribute to the prediction of output class as "1" -*i.e.*, P $\implies y_1 > y_2$. From figure 2, we have the weights from *X1* and *X2* to $N_{1,1}$ as 1 and -1, respectively while those to $N_{1,2}$ as 1 each. Therefore, the input vector takes the form (X_1 - X_2) for $N_{1,1}$ -*i.e.,* [1.X_1 + (-1).X_2] and the form (X_1 + X_2) for $N_{1,2}$ -*i.e.*, [1.X_1 + 1.X_2]. However, for the predicted class to be a 1, we require $N_{2,1}$ to be on while $N_{2,2}$ to be off. This is only true when $N_{1,1}$ is on and $N_{1,2}$ is off (since the weight of the connection $N_{1,1} \to N_{2,1}$ is positive while $N_{1,2} \to N_{2,2}$ is negative). Considering that the neurons employ ReLU activation, for $N_{1,1}$ to be positive, its input (X_1 - X_2) must be positive while for $N_{1,2}$ to be negative, its input (X_1 - X_2) must not be positive. Hence, the predicate $\sigma(X) = N_{1,1}(X) > 0 \wedge N_{1,2}(X) = 0$ holds for the postcondition that the predicted class is a '1' and amounts to the convex region (X_1 - X_2) > 0 $\wedge$ (X_1 + X_2) $\leq$ 0.

Proposition 2, on the other hand, can be derived from the fact that $N_{2,1}$ must be on and $N_{2,2}$ must be off for the above postcondition -*i.e.,* output = '1'. An input satisfying this pattern will have the output $[y_1, y_2] = [1.N_{2,1}(X) - 1.N_{2,2}(X), -1.N_{2,1}(X) + 1.N_{2,2}(X)]$ and will satisfy $y_1 > y_2$. This results in the pattern $\{N_{1,1} \to on$, $N_{1,2} \to off$, $N_{2,1} \to on$, $N_{2,2} \to off\}$ being an input property for P.

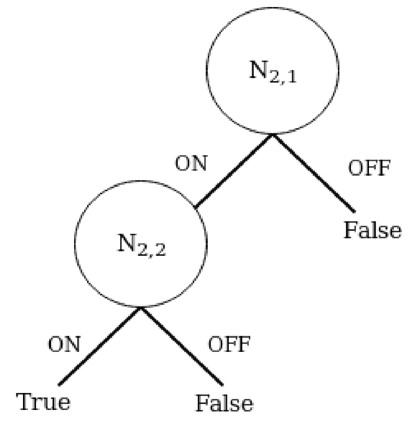
2 Computation of Network Properties

Technique 1: The first technique that the paper proposes for extraction of input properties is an *iterative relaxation* of decision patterns with an example of Reluplex [6]. The goal is to find the *minimal pattern* -*i.e.,* one with the least redundancy (see Section 1.2), σ that satisfies the postcondition *P* for all inputs. Algorithm 1 in the paper's appendix depicts the procedure for this by starting with a layer *l*, an input X whose output satisfies *P*, and keeps dropping constraints one at a time. First, the constraint $F(X) = y$ is dropped to mark that the output of an intermediate layer should not be necessarily equal to the final, and the decision process (DP) is invoked to check if $\sigma(X) \implies P(F(X))$. If this is false, it indicates that the input property is not adequate and so $\sigma(X) \wedge P(F(X))$ is returned as an input property. Otherwise, $\sigma(X)$ is considered to be an input property and we process further for minimality by dropping neurons of each layer one at a time. This step can be optimized by dropping all neurons of a layer starting from the last layer until we reach a critical layer where unconstraining all neurons invalidates the postcondition and hence, the neurons on this layer need to be unconstrained on a one-by-one basis to obtain a minimal subset of neurons.

Technique 2: A downside of the above approach is the expensive invocation of decision procedure (at least *n* + *m* calls where, *n* is the number of layers and *m* is the maximum number of neurons in a layer) at each step. In order to alleviate this, the paper proposes treating the extraction of properties as a *self-learning* technique. In specific, they use decision trees that leverage activation statuses of neurons in a layer to inference decision patterns. For a layer *l*, wrt which a postcondition *P* is to be learned, a dataset (see figure 3) is created using the two-step process: (a) the activation status (σ_X^l) of all its neurons are evaluated on an input X, (b) the boolean value representing whether the output F(X) satisfies P or not is then marked. A decision tree can then be built from this dataset wherein the path from a node to a leaf marked *True* denotes that the node satisfies the output property. In contrast, if a node has no such path, then the node is *impure* and can be pruned out (e.g., if the right branches off $N_{2,1}$ and $N_{2,2}$ in figure 3b had a node, it would be impure). The remaining nodes likely hold layer properties ("likely" because of the presence of nodes such as $N_{2,2}$ which might not satisfy the property based on the input) wrt postcondition and are ordered according to their support value.

$< x_1, x_2 >$	$< N_{1,1}, N_{1,2}, N_{2,1}, N2,2 >$	P(F(X))
(0,-1)	(on, off, on, off)	T
(0,1)	(off, on, off, on)	F
(1,0)	(on, on, on, off)	T
(1,-1)	(on, off, on, off)	T
(4,3)	(on, on, off, on)	F

(a) Extended training dataset



(b) Decision tree showing pattern for True: $\{N_{1,1} \rightarrow on, N_{1,2} \rightarrow on/off, N_{2,1} \rightarrow on, N_{2,2} \rightarrow off\}$

Figure 3. Figure showing an extension of example in the paper using decision tree learning for mining properties for nodes $N_{2,1}$ and $N_{2,2}$.

3 Applications

3.1 Analysis of ACASXU

The paper talks about two applications of the proposed inference technique. The first of these is the ACASXU, an unmanned collision avoidance system for aircraft. The authors analyze a DNN with 6 hidden layers, 50 ReLU activations and 384221 inputs with 5 labels. The input features comprise (1) Range: distance between ownship and intruder; (2)θ: angle of intruder relative to ownship heading direction; (3)ψ: heading angle of intruder relative to ownship heading direction; (4)v_{own}: speed of ownship; (5)v_{int}: speed of intruder; (6)τ: time until loss of vertical separation, and (7)α_{prev} previous advisory, while the labels consist of possible output actions that a ship might take: (0) Clear-of-Conflict (COC), (1) Weak Left, (2) Weak Right, (3) Strong Left, and (4) Strong Right.

The first step in the application involves inferring network properties wrt the postcondition that the output belongs to a certain label. This, in particular, proves the feasibility of extracting input and layer properties in terms of the on/off patterns of neurons in the network while also favoring *minimality* by showing that: (a) the patterns with lesser number of constrained neurons have higher support; (b) the layer properties have higher support than input properties.

The second step in the analysis involves explaining network behaviour based on the output. The authors use linear programming (LP) solver to calculate *under-approximation boxes* (i.e., solution intervals on each input dimension) for each of 3600 input properties. These are employed in finding the ranges for each input attribute which influenced the prediction - for example, 31900 $\leq$ range $\geq$ 37976, 1.684 $\leq \theta \leq$ 2.5133, ψ =-2.83, 414.3$\leq v_{own} \leq$ 506.86, v_{int}=300 predict correctly the decision to be COC. A further imposition of *minimality* on the input assignments as long as they satisfy the inferred properties makes it easier to prune irrelevant input features for a number of scenarios. This can therefore be compared to dimensionality reduction techniques used in standard machine learning.

The third and the last step uses layer patterns in simplifying difficult proofs of the form $A \Longrightarrow B$ where A specifies constraints on input attributes while B specifies an output turning advisory. The paper presents three such examples where the use of layer patterns as interpolants took as less as one-sixth of the time than for a direct proof in comparison to Reluplex.

3.2 Analysis of MNIST

The second analysis leverages the benchmark MNIST network[4] consisting of 60,000 training input images of 784 pixels each belong to 10 different classes (0 to 9) thus making its input space much larger than that of ACASXU. The property inference for MNIST reveals various such network properties which can serve as counter-examples for the input group which they belong to. Further, the explanation for network predictions uncover the visually similar nature of images that input properties capture in addition to the clusters of same digit written in different ways that the layer properties generate. The authors further select a case of misclassified input and compute an under-approximation box for it to show how many more similar inputs can be derived from the same box thus aiding in the study of the cause of misclassification. The method can be subsequently applied to cases such as the one shown in figure 1 as well as to safety-critical systems to find how the injection of noises outside the training distribution of a DNN might manipulate the model's decisions in favor of an adversary.

3.3 Distillation

The authors present distillation as a further application of how layer patterns can help save the computational requirements for large neural networks. Most of the large DNNs used for complex tasks have huge computational and memory requirement which might be infeasible for deployment to mobile devices under latency. For example, mobile camera apps extract facial features using DNNs for instant facial recognition while the user can still be taking the photo; it is therefore important that the model deployed by the app is responsive enough to do the same computations as a large network would do but without much lag in the accuracy. *Distillation* is based on the concept that a smaller model with fewer layers yet comparable accuracy can instead be used for deployment in such scenarios. The idea benefits from the fact that if the activation statuses of the neurons in a layer l satisfy the pattern $\boldsymbol{\sigma}^l$ then it implies that these neurons have a prediction capacity on par with the neurons in the further layers (since, satisfying $\boldsymbol{\sigma}^l$ means that the features extracted by these neurons are sufficiently detailed to be classified to a certain class c) and thus we can avoid processing the further layers of the network. Now, if the layer l lies far away from the final output layer, then distillation can result in elimination of thousands of mathematical operations during inference.[5]

The paper shows empirical evaluation of this approach on an eight-layered convolutional neural network for MNIST digit classification. Using layer patterns extracted by the decision tree method and applying a threshold on the accuracy score for the selection of these patterns, the paper reports distillation at the first and the second max pooling layer thus eliminating the execution of five and two layers respectively. While the degradation in accuracy is more from the first max pooling layer (0.9943 to 0.9903), the 22% saving on inference time is definitely an excellent trade-off. On the other hand, the decline in degradation of accuracy for second max pooling layer agrees with the trend of increase in complexity of features extracted with each layer.

4 Discussion

This article summarizes the work of Gopinath et al. (2019) which presents a white-box analysis of the behavior of neurons in DNNs. To this goal, the paper can thought of encompassing a range of other progress made recently. Ribeiro et al. (2018), for instance, compute *rules* based on black-box behavior of networks to represent local, "sufficient" conditions for predictions while Dhamdhere et al. (2018) use the notion of amount of information flowing through a neuron to understand its importance in predicting for a specific input, or over a set of inputs. While the former feature can be thought of having encapsulated by the *input properties* of the current work, the importance of individual neurons can be very well explained by looking at the *layer properties* they contribute to.

[4]MNIST database: http://yann.lecun.com/exdb/mnist/

[5]https://datascience.stackexchange.com/questions/24063/amount-of-multiplications-in-a-neural-network-model

A significant aspect of the work can be thought of its applications to safety-critical scenarios. For dependable neural networks, the conditional mapping between individual neurons to features, the use of sanitized data as well as formal analyses to validate correctness claims must be among the foremost design considerations [2]. While the current work covers the scope of the first and the third requirement, the validity of data is still a field to be extended. Given their vulnerability to adverserial perturbations, one possible solution is to use such formal methods to validate if the activations of the neurons follow a valid decision pattern or not - for example, a system should be robust enough to detect if the sticker placed by the banana (figure 1) has triggered a decision pattern different to thousands of other images of the banana alone, and manipulate its decisions accordingly. A similar idea has also been put forward in Daniel Casini's slides[6] which mentions the use of hard computing and redundant neural networks (trained on different data) to monitor anomalous behaviour followed by redirecting the decisions to safe actions.

Last but not least, the concept of distillation based on implied layer properties opens up a whole new world of optimization. Since the *adaptive* approach to distillation in the paper allows selective filtering of inputs for the large and small networks, this can have compelling applications for resource-constrained sensors where a device might prefer inferencing input within or send it over the cloud based on the available power and computation. Also, an interesting step would be to adapt multi-step knowledge distillation [8] used in teacher-student learning to see how compact a distilled network might be made.

References

Tom B. Brown, Dandelion Mané, Aurko Roy, Martín Abadi, and Justin Gilmer. Adversarial patch. *ArXiv*, abs/1712.09665, 2018.

Chih-Hong Cheng, Frederik Diehl, Gereon Hinz, Yassine Hamza, Georg Nührenberg, Markus Rickert, Harald Ruess, and Michael Truong-Le. Neural networks for safety-critical applications—challenges, experiments and perspectives. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1005–1006. IEEE, 2018.

Kedar Dhamdhere, Mukund Sundararajan, and Qiqi Yan. How important is a neuron. *ArXiv*, abs/1805.12233, 2018.

Divya Gopinath, Hayes Converse, Corina S. Pasareanu, and Ankur Taly. Property inference for deep neural networks. 2019.

Constance Heitmeyer. Developing safety-critical systems: the role of formal methods and tools. In *Proceedings of the 10th Australian workshop on Safety critical systems and software-Volume 55*, pages 95–99. Australian Computer Society, Inc., 2006.

Guy Katz, Clark W. Barrett, David L. Dill, Kyle Julian, and Mykel J. Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *CAV*, 2017.

Hiroshi Kuwajima, Hirotoshi Yasuoka, and Toshihiro Nakae. Open problems in engineering and quality assurance of safety critical machine learning systems. *arXiv preprint arXiv:1812.03057*, 2018.

Seyed-Iman Mirzadeh, Mehrdad Farajtabar, Ang Li, and Hassan Ghasemzadeh. Improved knowledge distillation via teacher assistant: Bridging the gap between student and teacher. *arXiv preprint arXiv:1902.03393*, 2019.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Anchors: High-precision model-agnostic explanations. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Zhi Quan Zhou and Liqun Sun. Metamorphic testing of driverless cars. *Commun. ACM*, 62:61–67, 2019.

[6] Slides 10-18: https://retis.sssup.it/~d.casini/papers/2018/slides/RTSOPS2018_DNN.pdf