Runtime Monitoring Neuron Activation Patterns

Chih-Hong Cheng*, Georg Nührenberg* and Hirotoshi Yasuoka†
*fortiss - Research Institute of the Free State of Bavaria
†DENSO CORPORATION
Email: {cheng,nuehrenberg}@fortiss.org, hirotoshi_yasuoka@denso.co.jp

***Abstract*—For using neural networks in safety critical domains, it is important to know if a decision made by a neural network is supported by prior similarities in training. We propose runtime neuron activation pattern monitoring - after the standard training process, one creates a monitor by feeding the training data to the network again in order to store the neuron activation patterns in abstract form. In operation, a classification decision over an input is further supplemented by examining if a pattern similar (measured by Hamming distance) to the generated pattern is contained in the monitor. If the monitor does not contain any pattern similar to the generated pattern, it raises a warning that the decision is not based on the training data. Our experiments show that, by adjusting the similarity-threshold for activation patterns, the monitors can report a significant portion of misclassfications to be not supported by training with a small false-positive rate, when evaluated on a test set.**

***Index Terms*—runtime monitoring, neural network, dependability, autonomous driving**

I. Introduction

For highly automated driving, neural networks are the *de facto* option for vision-based perception. Nevertheless, one fundamental challenge for using neural networks in such a safety-critical application is to understand if a trained neural network performs inference "*outside its comfort zone*". This appears when the network needs to significantly extrapolate from what it learns (or remembers) from the training data, as similar data has not appeared in the training process.

In this paper, we address this problem by *runtime monitoring neuron activation patterns*, where the underlying workflow is illustrated in Figure 1. After completing the training process, one records the neuron activation patterns for close-to-output neural network layers for all correctly predicted data used in the training process. Neurons in close-to-output layers in general represent high-level features, as demonstrated by recent approaches in interpreting neural networks [12]. As state-of-the-art neural networks commonly use ReLU or its variations as activation function, we select the ReLU *on-off activation pattern* to record the presence or absence of high-level features. At the same time, on-off patterns allow efficient storage using binary decision diagrams (BDDs) [1]. In operation, a classification decision is supplemented with a BDD-based monitor to detect whether the provided input has triggered an unseen neuron activation pattern - whenever an unseen activation pattern appears, the decision made by the neural network is considered to be less reliable. For the example in Figure 1-(b), the scooter is classified as a car, but as its neuron activation pattern is not among the existing patterns created from the training data, the monitor reports that

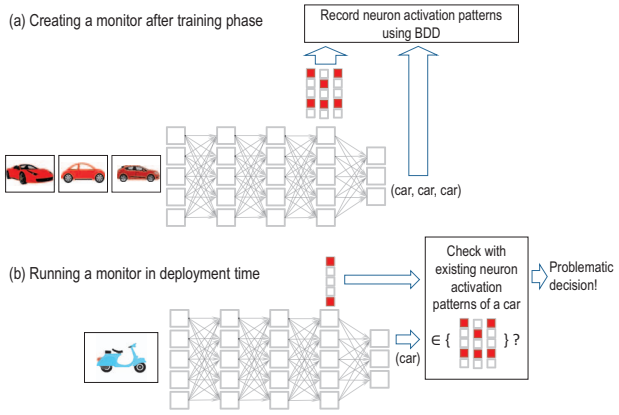

Fig. 1. High-level workflow on runtime monitoring neuron activation patterns.

the decision made by the neural network can be problematic. The frequent appearance of unseen patterns provides an indicator of data distribution shift to the development team; such information is helpful as it may indicate that a neural network deployed on an autonomous vehicle needs to be updated.

Nevertheless, for such an approach to be useful, we encounter technical difficulties where in the created monitor, the coarseness of abstraction should be *abstract enough, but not too abstract*. An illustration can be found in Figure 2, where given α to be all visited states from the training data, an abstraction such as α_1 allows nearly no generalization effect, making all encountered data in operation time to be "not visited". On the other hand, an abstraction such as α_3 is too coarse in that every observed pattern in operation time is identified to be "visited"; such a monitor is also not useful. Overall, we have applied the following techniques to control the coarseness of abstraction.

(Enlarge the abstraction) Apart from merely including visited patterns, we further develop technologies to enlarge the pattern space by considering all neuron activation patterns whose Hamming distance with existing patterns are within a certain threshold. It can also be efficiently implemented using existential quantification as commonly seen in many BDD software packages. Adding additional patterns does not influence performance - the membership query during runtime remains in the worst case in time linear to the number of neurons in the monitored layer (due to the use of BDDs). In addition, we apply gradient-based sensitivity analysis [9] to only monitor important neurons, thereby allowing unmonitored neurons to hold

978-3-9819263-2-3/DATE19/©2019 EDAA

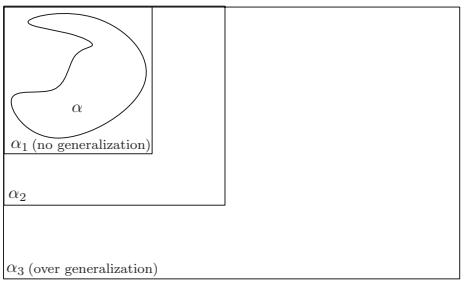

Fig. 2. Finding "just-right" abstraction for runtime monitors

arbitrary values in the abstraction. This also overcomes the limitation where the maximum number of BDD variables one can use in practice is around hundreds.

(Infer when to stop enlarging) To ensure that the abstraction is not too coarse, we take a validation set (which is expected to have the same distribution as in operation, but with ground-truth labels) and gradually increase the Hamming distance such that in the created region of abstraction, whenever the occurrence of out-of-pattern scenarios appears, it is also likely that misclassification appears. We applied this concept to decide the coarseness of abstraction for classifying standard image benchmarks such as MNIST [5] German Traffic Sign Recognition Benchmark (GTSRB) [10], as well as a vision-based front-car detector for automated highway piloting.

The rest of the paper is structured as follows. Section II describes how to build neuron activation pattern monitors with the use of BDDs. Section III gives examples in terms of controlling the coarseness of abstraction. We summarize related work in Section IV and conclude in Section V with further research directions.

II. Building Neuron Activation Pattern Monitors

We describe the underlying principles of our runtime monitoring approach for neural networks. For simplicity, the presented algorithm is for image classification, and we focus on runtime monitoring fully-connected neural network layers. Monitoring convolutional layers can be achieved by treating layers having convolutional filters as layers with fully connected neurons where missing connections are assigned with zero weights.

A neural network is comprised of L layers where operationally, the l-th layer for $l \in \{1, \ldots, L\}$ of the network is a function $g^{(l)} : \mathbb{R}^{d_{l-1}} \to \mathbb{R}^{d_l}$, with d_l being the dimension of layer l. Given an input $\mathsf{in} \in \mathbb{R}^{d_0}$, the output of the l-th layer of the neural network $f^{(l)}$ is given by the functional composition of the l-th layer and previous layers $f^{(l)}(\mathsf{in}) := \circ_{i=1}^{(l)} g^{(i)}(\mathsf{in}) = g^{(l)}(\ldots g^{(2)}(g^{(1)}(\mathsf{in})))$. For a neural network classifying C categories $d_L = C$. Given the computed output $f^{(L)}(\mathsf{in}) = (v_1, \ldots, v_C)$, the *decision* $\mathsf{dec}_{f^{(L)}}(\mathsf{in})$ of classifying input in to a certain class is based on choosing the index i with the maximum value v_i among elements in the output vector, i.e., $\mathsf{dec}_{f^{(L)}}(\mathsf{in}) := \mathsf{argmax}_{i \in \{1,\ldots,C\}}\{v_1, \ldots, v_C\}$.

An important case in modern neural networks is the use of layers implementing *Rectified Linear Unit* (ReLU), where the corresponding function $g^{(l)}$ maintains the input dimension and transforms an input vector element-wise by keeping its positive part, i.e., $g^{(l)}(v_1, \ldots, v_{d_{l-1}}) := (v'_1, \ldots, v'_{d_{l-1}})$ where $v'_i := \mathsf{max}(0, v_i)$ for $i \in \{1, \ldots, d_{l-1}\}$.

By interpreting an input element v_i to the ReLU layer as feature intensity, if v_i has value greater than zero, then it is considered to be *activated*, while v_i having value less or equal to zero is considered to be *suppressed* by ReLU. With this intuition in mind, our definition of a *neuron activation pattern* is based on capturing the activation and suppression of features.

*Definition 1 (**Neuron activation pattern**):* Given a neural network with input in and the l-th layer being ReLU, $\mathsf{pat}(f^{(l)}(\mathsf{in}))$, the neuron activation pattern at layer l, is defined as follows:

$$\mathsf{pat}(f^{(l)}(\mathsf{in})) := (p_{\mathrm{relu}}(v_1), \ldots, p_{\mathrm{relu}}(v_{d_l}))$$

where $(v_1, \ldots, v_{d_l}) = f^{(l)}(\mathsf{in})$ is the output from layer l, and $p_{\mathrm{relu}} : \mathbb{R} \to \{0, 1\}$ captures the activation cases:

$$p_{\mathrm{relu}}(x) = \begin{cases} 1 & x > 0 \\ 0 & otherwise \end{cases}$$

Let $\mathcal{T}$ denote the set of training inputs and let $\mathcal{T}_c \subseteq \mathcal{T}$ denote the set of all training images labelled as class c based on the ground truth. For each class c, we define the corresponding "comfort zone" for a neural network to be the set of activation patterns visited for all correctly classified training images, together with other neuron activation patterns that are close (via Hamming distance) to visited patterns.

*Definition 2 (**γ-comfort zone**):* Given a neural network and its training set $\mathcal{T}$, the γ-comfort zone $\mathcal{Z}_c^\gamma \subseteq \{0, 1\}^{d_l}$ for classifying class c, under the condition where the l-th layer is ReLU, is defined recursively as follows:

$$\mathcal{Z}_c^\gamma := \begin{cases} \{\mathsf{pat}(f^{(l)}(\mathsf{in})) \mid \mathsf{in} \in \mathcal{T}_c \wedge \mathsf{dec}_{f^{(L)}}(\mathsf{in}) = c\}, \text{if } \gamma = 0 \\ \mathcal{Z}_c^{\gamma-1} \cup \{\mathsf{p} \mid \mathsf{p} \in \{0,1\}^{d_l} \wedge \\ \qquad \exists \mathsf{p'} \in \mathcal{Z}_c^{\gamma-1} : \mathcal{H}(\mathsf{p}, \mathsf{p'}) = 1\}, \text{if } \gamma > 0 \end{cases}$$

where $\mathcal{H}(p, p')$ is the function to compute the Hamming distance between two pattern vectors $\mathsf{p}, \mathsf{p'} \in \{0, 1\}^{d_l}$.

Lastly, a neuron activation pattern monitor stores the computed comfort zone for each class using the training data.

*Definition 3 (**Neural activation pattern monitor**):* Given a neural network for classifying C classes, its training set and a user-specified γ, its neuron activation pattern monitor is defined as $\langle \mathcal{Z}_1^\gamma, \ldots, \mathcal{Z}_C^\gamma \rangle$.

Note that as $\mathcal{Z}_c^\gamma \subseteq \{0, 1\}^{d_l}$, the construction of $\mathcal{Z}_c^\gamma$ can be done using binary decision diagrams with d_l variables. Algorithm 1 describes how to construct such a monitor, where $\mathsf{bdd.emptySet}$, $\mathsf{bdd.or}$, and $\mathsf{bdd.encode}$ are functions used to create an empty set, to perform set union, and to encode an activation pattern into BDDs. The function $\mathsf{bdd.exists}(j, \mathsf{set})$ performs the existential quantification on set over the j-th variable.

In Algorithm 1, lines 4 to 8 record all visited patterns to form $\mathcal{Z}_c^0$. Subsequently, lines 9 to 14 build $\mathcal{Z}_c^i$ from $\mathcal{Z}_c^{i-1}$. In

Input: neural network and l-th layer to monitor, training set $\mathcal{T}$, user specified γ

Output: runtime activation pattern monitor $\langle \mathcal{Z}_1^\gamma, \ldots, \mathcal{Z}_C^\gamma \rangle$

/* initialize monitors as empty BDDs */

1 **for** $c \in C$ **do**

2 $\quad \mathcal{Z}_c^0 \leftarrow$ bdd.emptySet()

3 **end**

/* iterate all images */

4 **for** $in \in \mathcal{T}$ **do**

/* check if prediction is correct */

5 $\quad$ **if** $dec_{f^{(L)}}(in) = c \wedge in \in \mathcal{T}_c$ **then**

/* add activation pattern to the corresponding BDD */

6 $\quad\quad \mathcal{Z}_c^0 \leftarrow$ bdd.or($\mathcal{Z}_c^0$, bdd.encode(pat($f^{(l)}$(in))))

7 $\quad$ **end**

8 **end**

9 **for** $c = 1, \ldots, C,\ i = 1, \ldots, \gamma$ **do**
$\quad \mathcal{Z}_c^i \leftarrow$ bdd.emptySet();

10 **for** $c = 1, \ldots, C$ **do**

11 $\quad$ **for** $i = 1, \ldots, \gamma$ **do**

12 $\quad\quad$ **for** $j = 1, \ldots, d_l$ **do**
$\quad\quad\quad \mathcal{Z}_c^i \leftarrow$ bdd.or($\mathcal{Z}_c^i$, bdd.exists($j, \mathcal{Z}_c^{i-1}$)) ;

13 $\quad$ **end**

14 **end**

15 **return** $\langle \mathcal{Z}_1^\gamma, \ldots, \mathcal{Z}_C^\gamma \rangle$

Algorithm 1: Building a neuron activation pattern monitor after training

particular, computing the enlarged $\mathcal{Z}_c^\gamma$ from $\mathcal{Z}_c^{\gamma-1}$ can be efficiently achieved using the existential quantification operation as listed in line 12. Consider an example where $\mathcal{Z}_c^0 = \{001\}$, then the operation bdd.exists$(j, \mathcal{Z}_c^0)$, for $j = 1, 2, 3$, creates $\{$-01$\}, \{$0-1$\}, \{$00-$\}$ respectively. The union over existentially quantified result creates an enlarged set containing additional patterns with Hamming distance equal to 1.

(Neuron selection via gradient analysis) For layers with large neuron amounts, as the use of BDD has practical variable limits around 200, one extension is to only monitor the activation patterns over a subset of neurons that are important for the classification decision. One way of selecting neurons to be monitored is to apply gradient-based sensitivity analysis similar to the work of saliency map [9]. The underlying principle is that for the output of neuron n_i over neuron n_c producing output class c, one computes $\frac{\partial n_c}{\partial n_i}$. Subsequently, one only selects neuron n_i if $|\frac{\partial n_c}{\partial n_i}|$ is large, as the change of value n_i significantly influences the output c due to the derivative term.

As a special case, if one monitors patterns over the neuron layer immediately before the output layer, and there is no non-linear activation in the output layer (which is commonly seen in practice), $\frac{\partial n_c}{\partial n_i}$ is simply the weight connecting n_i to n_c.

III. Controlling the Abstraction

As stated in the introduction, the coarseness of abstraction should be carefully designed to make the resulting monitor useful. Both the number of neurons being monitored and the value γ are hyper-parameters to control the coarseness of abstraction. We have implemented the concept to examine the effect of different γ using the PyTorch machine learning framework and the python-based BDD package `dd`[1].

ID	Classifier	Model architecture	Accuracy (train/validation)
1	MNIST	ReLU(Conv(40)), MaxPool, ReLU(Conv(20)), MaxPool, ReLU(fc(320)), ReLU(fc(160)), ReLU(fc(80)), **ReLU(fc(40))**, fc(10)	99.34%, 98.81%
2	GTSRB	ReLU(BN(Conv(40))), MaxPool, ReLU(BN(Conv(20))), MaxPool, ReLU(fc(240)), **ReLU(fc(84))**, fc(43)	99.98%, 96.73%

TABLE I
Architectures and accuracies of the networks used in the experiment. Convolutional layers (Conv) have kernel size $(5, 5)$ and stride $(1, 1)$. We use 2×2 max pooling layers (MaxPool). Fully-connected layers and batch normalization are denoted by fc$(\cdot)$ and BN$(\cdot)$. The layer being monitored is highlighted in bold text.

ID	misclassification rate	γ	$\frac{\#\textit{out-of-pattern images}}{\#\textit{total images}}$	$\frac{\#\textit{out-of-pattern misclassified images}}{\#\textit{out-of-pattern images}}$
1	1.19%	0	7.66%	10.70%
		1	2.01%	21.89%
		2	0.6%	31.66%
2	3.27%	0	32.92%	10.13%
		1	15.0%	19.44%
		2	7.08%	41.17%
		3	4.58%	54.54%

TABLE II
Results of applying runtime neuron activation monitoring

Based on two publicly available image classification datasets MNIST [5] and GTSRB [10], we trained two neural networks. The architectures of the networks are summarized in Table I. After training, we build the runtime monitors based on Algorithm 1. For network 2, in the experiment we (i) only construct the monitor for the stop sign ($c = 14$) and (ii) out of 84 neurons in a layer, only 25% are monitored based on gradient-based analysis. We have gradually increased γ and recorded the rate of out-of-pattern images for all validation images, as well as the portion of misclassified images within out-of-pattern images.

For network 1 classifying MNIST, the rates of $\frac{\#\textit{out-of-pattern images}}{\#\textit{total images}}$ for all $\gamma \in \{0, 1, 2\}$ are all relatively small. For network 2 classifying GTSRB, one can argue that the abstraction using $\gamma = 0$ is not coarse enough, as the network has a low mis-classification rate (around 3.27%) but the monitor reports that around 32.92% of the images create patterns that are not included in the monitor.

(MNIST with $\gamma = 2$) If there is no distributional shift in operation, the monitor will not signal problems in 99.4% (100% − 0.6%) of its overall operation time, implying that it is largely silent. Nevertheless, whenever it signals an issue of unseen patterns, apart from arguing that the network is making a decision without prior similarities,

[1] `dd`: https://pypi.org/project/dd/

one may even argue that there is a non-neglectable probability of 31.66% where the decision being made by the network is problematic[2], although the neural network may still report that the input is classified to the class with a high probability.

(GTSRB with $\gamma = 3$) If there is no distributional shift in operation, the monitor will not signal problem in 95.42% ($100\% - 4.58\%$) of its operating time. Whenever it signals an issue of unseen patterns, there is a non-neglectable probability of 54.54% where it is indeed misclassified.

(Case Study) We also experimented the runtime monitoring technique on a vision-based front-car detection system for highway piloting. The vision subsystem (cf. Figure 3) contains three components: (1) vehicle detection, (2) lane detection, and (3) front-car selection. The front-car selection unit is implemented using a neural network-based classifier, which takes the lane information and the bounding box of vehicles, and produces either an index of the bounding vehicle or a special class "♯" for which no forward vehicle is considered to be a front car.

IV. Related Work

Using neural networks in safety critical applications has raised needs for creating dependability claims. Recent results in compile-time formal verification techniques such as RuLUplex [4] or Planet [2] use constraint solving to examine if for all inputs within a bounded polyhedron, it is possible for the network to generate undersired outputs. These techniques are used when a risk property is provided by domain experts beforehand, and they are only limited to piecewise linear networks with a few number of neurons. Our work of neuron monitoring is more related to the concept of runtime verification [6], which examines if a runtime trace has violated a given property. The generalizability condition, as defined by the γ-comfort zone created after training, can be understood as a safety property. To the best of our knowledge, we are unaware of any work in runtime verification that considers the problem of generalizability monitoring of neural networks. In terms of scalability, our framework also allows taking arbitrary large networks with other nonlinear activation functions, so long as the neurons being monitored are ReLU.

Lastly, within machine learning (ML), the work of filtering adversarial attacks [3], [11] reply on creating another ML component to perform detection (thus preventing the network from making wrong decisions). Our proposed method differs from these ML-based approach in that the *sound* over-approximation of the visited inputs implies that if the monitor reports the occurrence of an unseen pattern, the occurrence is always genuine. The *sure guarantee* (in contrast to concepts such as *almost-sure*[3] which is the best one can derive with statistical machine learning methods) makes the certification of such a monitor in the safety domain relatively easier. In particular within the domain of autonomous driving, it is highly likely that the test set used in engineering time will deviate from the real world data (the black swan effect), making any probabilistic claim hard to be certified.

[2]The argument is based on an assumption where no distributional shift implies that $\frac{\#\textit{out-of-pattern misclassified images}}{\#\textit{out-of-pattern images}}$ remains the same in validation and in operation.

[3]https://en.wikipedia.org/wiki/Almost_surely

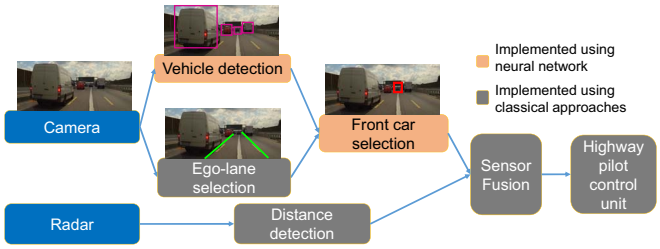

Fig. 3. High-level architecture of a front-car detection unit for a highway piloting system.

V. Concluding Remarks

In this paper, we proposed neuron activation pattern monitoring as a method to detect if a decision made by a neural network is not supported by prior similarities in training. We envision that a neuron activation pattern monitor can be served as a medium to assist the sensor fusion process on the architecture level, as a decision made by the network may not be fully trusted due to no ground-truth being offered in operation time.

The established connection between formal methods and machine learning also reveals several possible extension schemes. (1) The technique shall be directly applicable on object detection networks such as YOLO [8], whose underlying principle is to partition an image to a finite grid, with each cell in the grid offering object proposals. (2) We are also studying the feasibility on more refined domains using tools such as difference bound matrices [7], in order to better capture an abstract representation of the visited activation patterns.

References

[1] R. E. Bryant. Symbolic boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.

[2] R. Ehlers. Formal verification of piece-wise linear feed-forward neural networks. In *ATVA*, pages 269–286. Springer, 2017.

[3] K. Grosse, P. Manoharan, N. Papernot, M. Backes, and P. McDaniel. On the (statistical) detection of adversarial examples. *arXiv:1702.06280*, 2017.

[4] G. Katz, C. Barrett, D. L. Dill, K. Julian, and M. J. Kochenderfer. Reluplex: An efficient smt solver for verifying deep neural networks. In *CAV*, pages 97–117. Springer, 2017.

[5] Y. LeCun. The MNIST database of handwritten digits. *http://yann.lecun.com/exdb/mnist/*, 1998.

[6] M. Leucker and C. Schallhart. A brief account of runtime verification. *The Journal of Logic and Algebraic Programming*, 78(5):293–303, 2009.

[7] A. Miné. A new numerical abstract domain based on difference-bound matrices. In *Programs as Data Objects*, pages 155–172. Springer, 2001.

[8] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *CPVR*, pages 779–788, 2016.

[9] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv:1312.6034*, 2013.

[10] J. Stallkamp, M. Schlipsing, J. Salmen, and C. Igel. The german traffic sign recognition benchmark: a multi-class classification competition. In *IJCNN*, pages 1453–1460. IEEE, 2011.

[11] W. Xu, D. Evans, and Y. Qi. Feature squeezing: Detecting adversarial examples in deep neural networks. *arXiv:1704.01155*, 2017.

[12] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *ECCV*, pages 818–833. Springer, 2014.