Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa

A numerical verification method for multi-class feed-forward neural networks

Daniel Grimm [a], Dávid Tollner [b], David Kraus [a], Árpád Török [b,*], Eric Sax [a], Zsolt Szalay [b]

[a] *Institut fuer Technik der Informationsverarbeitung, Karlsruhe Institute of Technology, Engesserstr. 5, Karlsruhe, 76131, Germany*
[b] *Department of Automotive Technologies, Faculty of Transportation Engineering and Vehicle Engineering, Budapest University of Technology and Economics, Műegyetem rkp. 3., Budapest, 1111, Hungary*

ARTICLE INFO

Keywords:
Neural network verification
Nonlinear optimization
Explainable neural networks

ABSTRACT

The use of neural networks in embedded systems is becoming increasingly common, but these systems often operate in safety–critical environments, where a failure or incorrect output can have serious consequences. Therefore, it is essential to verify the expected operation of neural networks before deploying them in such settings. In this publication, we present a novel approach for verifying the correctness of these networks using a nonlinear equation system under the assumption of closed-form activation functions. Our method is able to accurately predict the output of the network for given specification intervals, providing a valuable tool for ensuring the reliability and safety of neural networks in embedded systems.

1. Introduction

In many areas of critical infrastructures, such as Industry 4.0, the automotive sector or energy technology, researchers are studying the application of neural networks (NNs). The most important categories of tasks performed by NNs are classification and regression. Based on training, testing and validation data, the networks can be developed, and their accuracy determined on the known data, e.g. Struye and Latré (2020). However, what happens in practical use for previously unknown data? Because of this issue, artificial intelligence (AI)-based systems remain black-boxes from a societal perspective, with little trust placed in them. However, especially in the area of critical applications, the safety and security of deployed systems is the most important constraint, e.g. Mekonnen and Sipos (2022). In order to be able to justify the use of AI-based systems in practice, trust in the safety and security of the systems must therefore be created, e.g. as researched by Vu and Lim (2022). Therefore, the research field of verification and validation of AI-based systems is flourishing. Some examples are in the field of automated driving, e.g. Ehlers (2017a) and Vishnukumar, Butting, Müller, and Sax (2017), or aircraft automation, e.g. Katz, Barrett, Dill, Julian, and Kochenderfer (2017). Especially for NNs, various researchers investigate methods to evaluate their robustness, or correctness. More often, this boils down to modeling the problem as a system of linear or nonlinear equations and solving them using methods from linear or nonlinear programming. Depending on the type and size of the neural network, approximations and transformations are required for modeling the problem. Many works focus on the verification of Rectified Linear Unit (ReLU) activation function networks, e.g. Katz et al. (2017). As a nonlinear activation function that is not continuously differentiable, NNs using this activation function cannot be readily analyzed using methods that require a continuous derivative. Therefore, an active area of research is to transform networks with ReLU activation function so that they are solvable by conventional optimization methods. Some noteworthy publications were made by Brown, Schmerling, Azizan, and Pavone (2022), Bunel, Turkaslan, Torr, Kohli, and Mudigonda (2018), and Luenberger, Ye, et al. (1984). This work continues a previous work of our research group's collaboration, in which a novel approach was developed to evaluate the correctness of feed-forward NNs in binary classification problems. In this previous work of Tollner, Ziyu, Zöldy, and Török (2022), the ReLU activation function of the network was replaced by the *mReLU* (modified ReLU) function, which is continuous differentiable. Thus, after training the network on an artificial data set, the correctness of the network's prediction in a specified interval can be evaluated by solving the equivalent system of nonlinear equations. In the work at hand, we extend the approach to the analysis of multi-class NNs. For this purpose, we present how the problem has to be reformulated. First, we show how the Softmax activation function can be used in our framework. Furthermore, the work indicates that comparable results can also be obtained with a mapping of the multi-class problem to a Sigmoid activation

* Corresponding author.
E-mail addresses: daniel.grimm@kit.edu (D. Grimm), tollner.david@kjk.bme.hu (D. Tollner), d.kraus@kit.edu (D. Kraus), torok.arpad@kjk.bme.hu (Á. Török), eric.sax@kit.edu (E. Sax), szalay.zsolt@kjk.bme.hu (Z. Szalay).

https://doi.org/10.1016/j.eswa.2024.123345
Received 20 June 2023; Received in revised form 16 January 2024; Accepted 23 January 2024
Available online 28 January 2024

function. A benefit of this approach is that the optimization procedure only has to be carried out once. Additionally, an investigation of the Swish activation function, analyzed by Ramachandran, Zoph, and Le (2017), as an alternative to ReLU or *mReLU*, is implemented. In the study of Ramachandran et al. (2017) it has been shown that the Swish activation function always yields as good or better results as the ReLU function. However, the advantage in comparison to ReLU is that this function is continuously differentiable.

2. Motivation and objectives

In order to ensure the safety, reliability, and acceptable performance of highly automated vehicle systems, it is crucial for the industry to provide guarantees for the operation of Neural Networks applied in automotive systems. Highly automated vehicles widely use NNs for tasks such as object detection, lane keeping, and decision-making. The functions are often involved in providing passenger or pedestrian safety. Validating and verifying NNs help to identify and mitigate potential risks and vulnerabilities in the models, reducing the likelihood of severe accidents or malfunctions. At the same time, the automotive industry is regulated by strict standards to warrant the safety of vehicles on the road. Compliance with these standards is mandatory, and validation and verification processes help demonstrate that the NNs meet the required safety and performance standards. Neural networks applied by the automotive industry must perform effectively in diverse and unpredictable real-world conditions. Considering Barrachina, Boldizsar, Zoldy, and Torok (2019)'s results, NNs cannot provide solutions to all emerging problems in the industry, thus it is important to continuously develop verification solutions and improve their efficiency. Validating and verifying NNs can provide guarantees how well the developed neural network can operate in different circumstances/environments, weather conditions, and traffic scenarios. On the other hand, NNs must be resilient to failures and other unexpected events. Validation and verification of NNs contribute to identifying weaknesses and vulnerabilities that may lead to hazardous events. Beyond this, artificial intelligence based systems in highly automated vehicles are often subjected to continuous updates and improvements. Iteratively and continuously validating and verifying neural network based decision processes contribute to maintaining or enhancing system safety and performance over time. In summary, neural network validation and verification in the automotive industry is a critical task to warrant the safe and reliable deployment of artificial intelligence based functions in highly automated vehicles. It has to include comprehensive testing and assessment of diverse scenarios, and compliance with industry standards and regulations.

The most important scientific target related to neural network validation and verification in the automotive industry is to introduce well-funded, approved, efficient and well-applicable methodologies providing a high level of safety assurance. As Cao and Zoldy (2021) stated, safety is of key importance in the development and deployment of highly automated vehicles, where artificial intelligence based solutions play a crucial role in decision-making and control systems. Ensuring that these networks are reliable, robust, and free from critical errors is essential to prevent accidents and protect human lives. In the case of safety–critical vehicle functions, it is crucial to provide full explainability and transparency for NNs. For this, it is indispensable to improve the interpretability of neural network models to enhance understanding of their decision-making processes. This is critical for discovering errors, assessing system behavior, and building trust among users, policy makers, and users/human participants. Overall, the scientific gap in this domain is the lack of a comprehensive framework for validating and verifying NNs in highly automated vehicles, with a primary focus on safety. Achieving this target requires interdisciplinary collaboration among experts in machine learning, automotive engineering, control systems to create safe, efficient and reliable artificial intelligence based systems for the automotive industry. In more detail, it can be concluded that some of the formal efficient methods can be used under strict conditions, such as monotonicity or piecewise linearity of the activation function. This is a strong constraint that limits the applicability of these models. Another important challenge is to avoid the use of simplified models whenever possible and to examine the real model directly.

3. Contribution

The main contribution of our research results is that the developed framework provides a comprehensive, general-purpose solution for the verification of neural networks, with the potential for application in safety–critical systems. The new method opens up the possibility to test neural networks built from activation functions for which the conditions of monotonicity or piecewise linearity are not fulfilled. Another important research result is that the method tests the real model structure, rather than a simplified model describing the process, which would only provide approximate results due to the simplification or would require an exponentially large number of tasks.

4. Background

Many activation functions are used in the field of neural networks, with advantages and disadvantages. The (non-linear) activation function is needed because all neurons would provide linear transformation of the inputs without it. The linear mapping is additive and homogeneous. Thus, NNs with only linear activation functions can be reduced to a single neuron. This neuron would only be able to create a linear decision boundary, so it would be able to solve only linearly separable problems. In summary, the (nonlinear) activation function adds the nonlinearity to the neural network. The continuously differentiable property is also necessary for enabling gradient-based optimization algorithms. There is no universal function that always performs better than the others.

4.1. Linear activation functions

Rosenblatt (1958)'s perception is one of the first and simplest NNs, a single-layer feed-forward network with one neuron. In this case, the original activation was the Signum function for all $x \in \mathbb{R}$ (Eq. (1)):

$$y_{sgn}(x) = \begin{cases} -1, & \forall x < 0 \\ 0, & x = 0 \\ 1, & \forall x > 0 \end{cases} \tag{1}$$

The Signum function is only used as a regularization technique in modern neural network applications to have smaller weights and to prevent overfitting. A commonly used version is the Binary step function (Fig. 1), where the output is 0 or 1, depending on whether the output is positive or negative. In this case, the point $x = 0$ can be arbitrarily assigned to 0 or 1. The neurons can be interpreted as deactivated or activated with these outputs, respectively.

The Binary step function has only two outputs, so it is widely used for binary classification problems. If the problem is linear, one neuron is sufficient. In the nonlinear case, the hidden activation functions can be arbitrary, and the step function is used in the output neuron, so only one neuron is needed in the output layer.

The problem with the Signum and the Binary step function is that they are not differentiable at the point $x = 0$, therefore the gradient is undefined, in theory. But in practice, the derivative can be forced to be zero, as at all other points. Thus, the derivative is constant, independent of the input, and ruins the logic of the backpropagation algorithm.

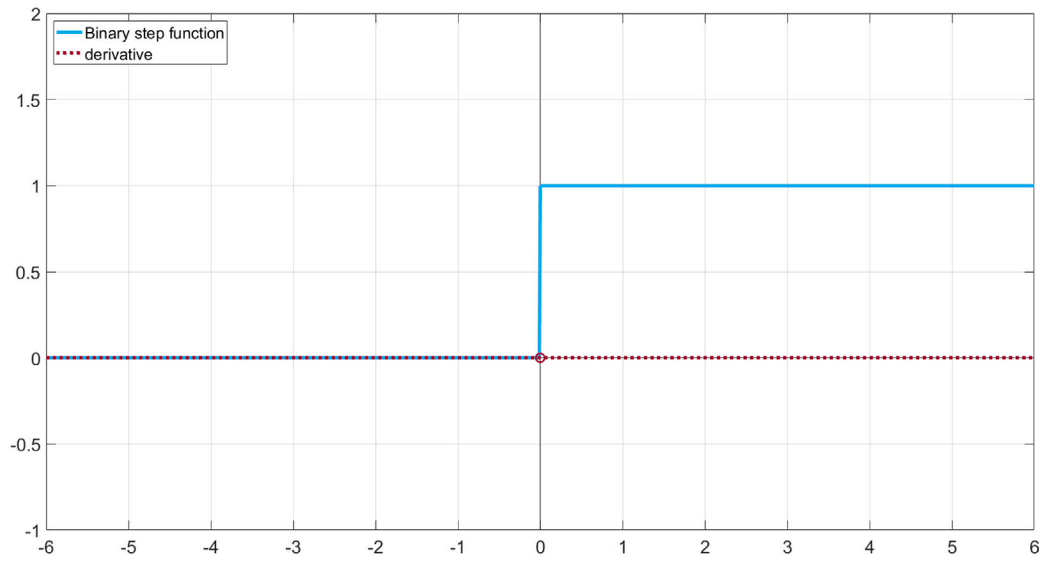


Fig. 1. The shape of the Binary step activation function and its derivative.

4.2. S-shaped non-linear activation functions

The composition of non-linear activation functions allows back-propagation and adds nonlinearity to the system. Moreover, as the Universal approximation theorem states, any Borel measurable function can be approximated to any desired degree of accuracy with only one hidden layer network using squashing activation functions (Hornik, Stinchcombe, & White, 1989).

Logistic and Sigmoid function

A traditional activation function is the Logistic function (Eq. (2)), which is also often used in population theory.

$$y_{logi}(x) = \frac{L}{1 + e^{-k(x-x_0)}} \tag{2}$$

where L is the supremum of the curve, k is the logistic growth rate and x_0 is x-value of the function's midpoint. The Sigmoid function is continuous, differentiable and monotonic.

Many hyperparameters can be set when training NNs, so parameterizing the activation function makes the task much more complicated. Instead, the Sigmoid function is commonly used, which is the special case of the Logistic function where $L = 1$, $k = 1$ and $x_0 = 0$. The Sigmoid is an S-shaped curve, which is bounded in the range of $(0, 1)$ (Fig. 2). This output can be interpreted as probability. In addition, it takes values close to its lower and upper bounds in a relatively small input interval. Due to these two properties, it is often used as an activation function in the output layer to solve 2-class classification problems. As it can be seen, the function has a nonzero-mean output. This can be a problem because all the neurons will have the same sign, which makes the training of the network more difficult and unstable and also leads to slower convergence (Amari, 1998).

The Logistic function and its derivative are smooth. The gradient value approaches zero at both positive and negative ends, therefore the function suffers from the vanishing gradient problem. This significantly impacts performance because if the gradient vanishes, the weights cannot be updated.

Due to the exponential calculations, the Logistic function is computationally expensive. To compensate for this, the gradient calculation is straightforward (Eq. (3)), which can speed up training.

$$\left(y_{sgn}(x)\right)' = y_{sgn}(x)\left(1 - y_{sgn}(x)\right) \tag{3}$$

Hyperbolic tangent function

The Hyperbolic tangent function – also known as the Tanh function – can be a good alternative to replace the Logistic function. Mathematically, it can represented as (Eq. (4)):

$$y_{tanh}(x) = \frac{1 - e^{-x}}{1 + e^{-x}} \tag{4}$$

Tanh and Sigmoid functions look similar (Fig. 3); however, a significant difference is that Tanh crosses the origin, which can lead to faster convergence. Also, it can be seen that the gradients do not decrease quickly like the Sigmoid functions. Therefore, the gradient does not vanish rapidly.

The connection between the Tanh and the Sigmoid functions is shown in Eq. (5). The Tanh contains all the properties of Sigmoid with the further advantage of crossing the origin. Therefore, it can provide better performance results.

$$y_{tanh}(x) = 2\, y_{sgn}(2\, x) - 1 \tag{5}$$

4.3. Piecewise linear activation functions

A different type of non-linear function is when the activation function is a combination of two linear functions.

ReLU function

The Rectified Linear Unit (ReLU) function defined by Nair and Hinton (2010) (Eq. (6)) is one of the most used functions in the hidden layers, as it eliminates several disadvantages of S-shaped curves. Although, because of the linear functions, it does not add any true nonlinearity to the system.

$$y_{relu}(x) = \max\{0, x\} \tag{6}$$

ReLU is a continuous, monotonic function and bounded from below, however it is not differentiable at $x = 0$. In practice, the gradient is defined as 0 at this point (Fig. 4). ReLU is not bounded from above, so it does not suffer from the vanishing gradient problem.

ReLU only activates the neuron if its input is positive. Therefore, many neurons are not involved in learning, reducing the tendency to overfitting and increasing the computational efficiency. Also, it is faster to calculate the values of ReLU and its derivative compared to S-shaped functions, further reducing computational costs.

However, ReLU has a major drawback. If the input is negative, then the gradient will be zero. Therefore, there will be no change in weights during training; this is the dying neuron or dying ReLU problem.

Leaky ReLU, Parametric ReLU

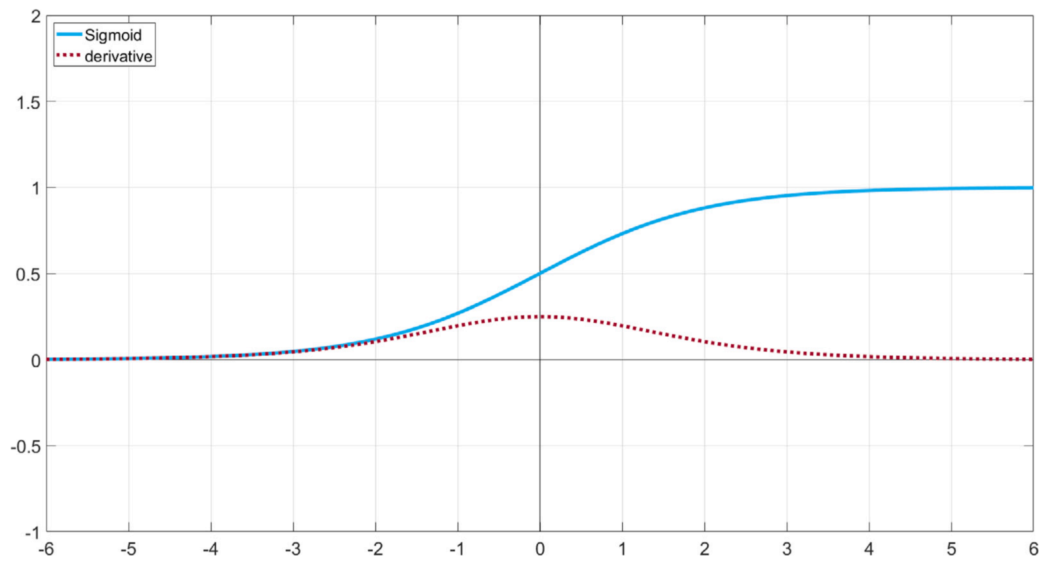


Fig. 2. The shape of the Sigmoid activation function and its derivative.

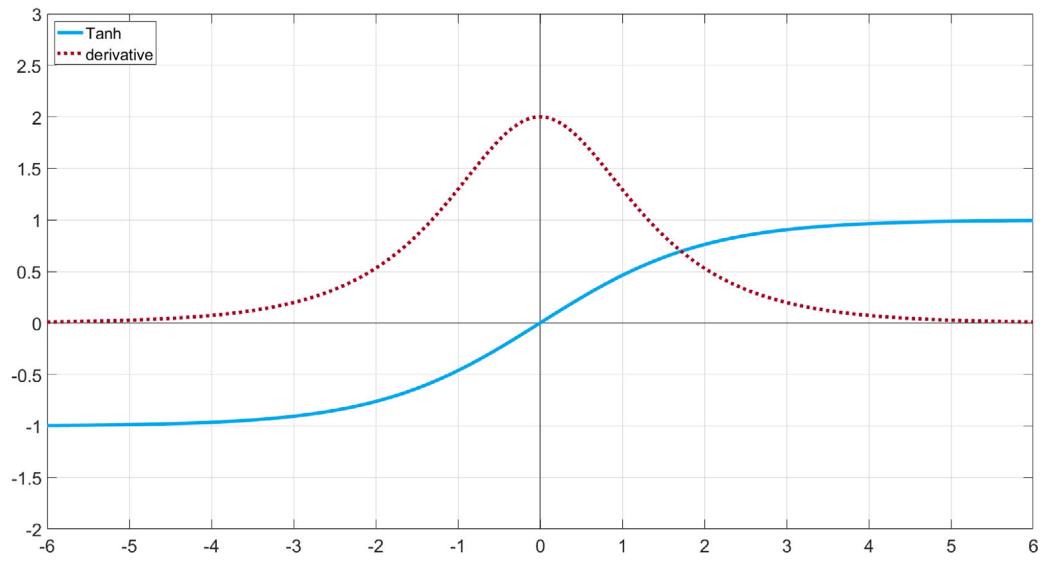


Fig. 3. The shape of the Tanh activation function and its derivative.

To overcome the dying neuron problem, Leaky ReLU introduced by Maas, Hannun, and Ng (2013) assigns a positive slope linear function to the negative inputs (Eq. (7)). For these inputs, the gradient will be 0.01 instead of 0 (Fig. 5). So, these parameters will also be updated.

$$y_{lrelu}(x) = \max\{0.01\,x, x\} \tag{7}$$

The Parametric ReLU can be seen as an extension of Leaky ReLU, where the 0.01 constant is replaced by a parameter (Eq. (8)).

$$y_{prelu}(x) = \max\{\alpha\,x, x\} \tag{8}$$

where α is a positive constant, typically $0.01 \leq \alpha \leq 0.1$.

In addition to the advantages, an extra hyperparameter is added to the system. This α value can be data-specific; its value needs to be set properly, as the network is sensitive to it. Although, the α can also be a learnable parameter, which is learned along with the network weights during the training phase.

4.4. Other activation functions

As neural networks have evolved, new activation functions have appeared in recent years. These functions were designed to combine the advantages of ReLU and Sigmoid functions and eliminate their drawbacks. We have collected these properties and their explanations in a list, which makes the analysis of the functions easier. The following functions are smooth, continuously differentiable, non-linear, and bounded from below (except Softmax).

Advantages:

+ **Smooth:** preventing jumps in output values, accelerates optimization and helps generalization
+ **Differentiable:** enabling gradient-based optimization algorithms
+ **Non-linear:** helping to learn complex patterns
+ **Bounded from below:** deactivating neurons, sparsing the network
+ **Zero centered:** faster convergence
+ **Negative output:** improving the ability to learn

Disadvantages:

– **Bounded (from above and below):** vanishing gradient problem
– **Exponential term:** computationally expensive

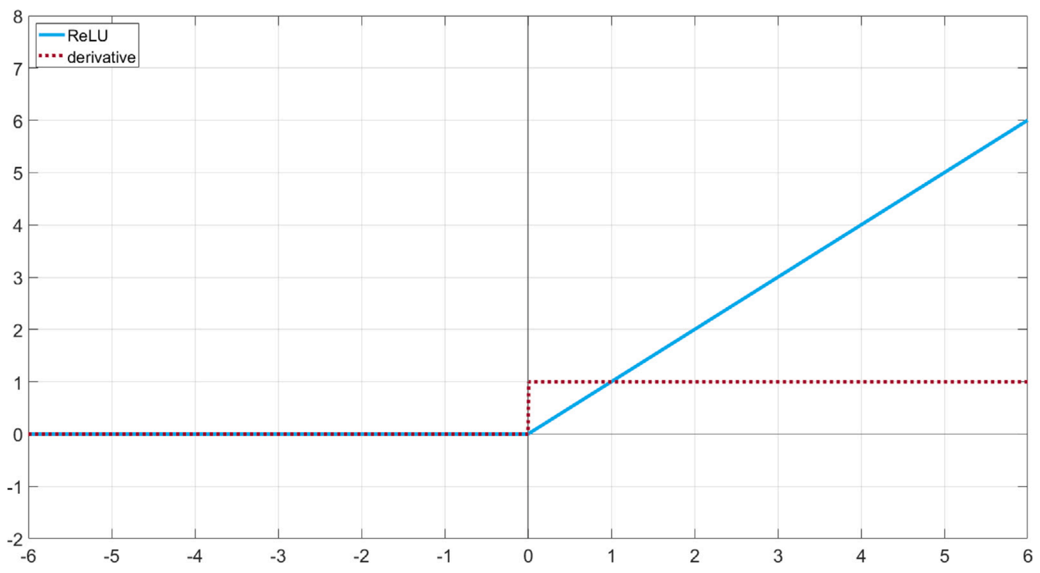


Fig. 4. The shape of the ReLU activation function and its derivative.

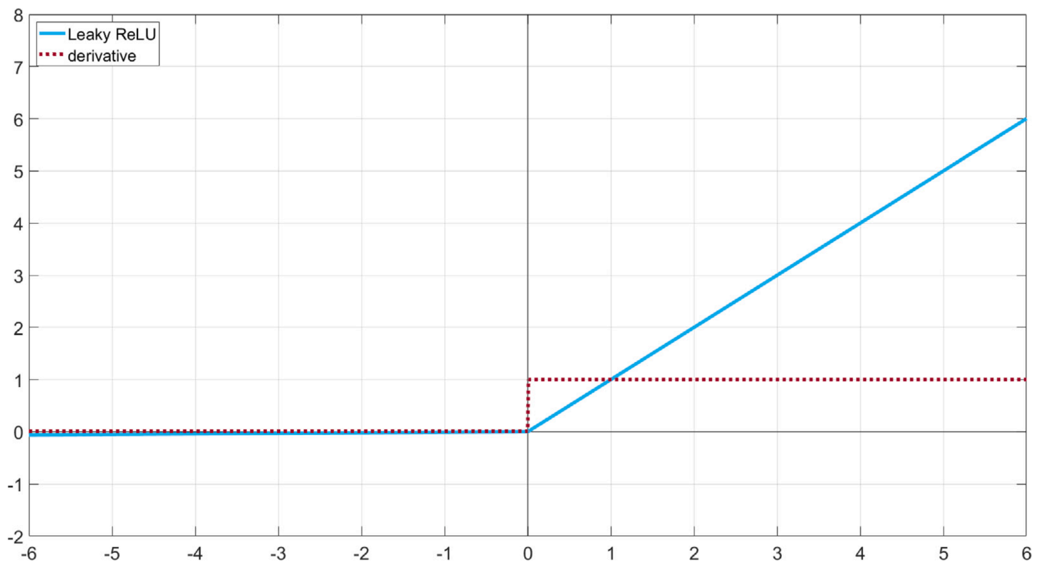


Fig. 5. The shape of the Leaky ReLU activation function and its derivative.

– **0 gradients:** dying ReLU problem
– **>1 gradients:** exploding gradient problem
– **Hyperparameter:** setting its value correctly is difficult

Softplus

Dugas, Bengio, Bélisle, Nadeau, and Garcia (2000) presented the Softplus activation function (Eq. (9)), which is a smoothed version of ReLU. Unfortunately, this also contains an exponential term.

$$y_{softp}(x) = \ln(1 + e^x) \tag{9}$$

The relationship between Softplus and the Sigmoid function is shown by (Eq. (10)):

$$\left(y_{softp}(x)\right)' = \frac{e^x}{1 + e^x} = \frac{1}{1 + e^{-x}} = y_{sgn}(x) \tag{10}$$

Swish

The Swish function was introduced in 2017 by Ramachandran et al. (2017). This function combines the advantages of Sigmoid and ReLU.

$$y_{swish}(x) = x\, y_{sgn}(x) = \frac{x}{1 + e^{-x}} \tag{11}$$

Swish, in addition to the abovementioned properties, is a non-monotonic function (Fig. 6). However, one negative property of the sigmoid function is preserved: the computation cost. Furthermore, this function is self-regularized, meaning the output approaches zero for large negative inputs. At the same time, there is a negative bump for small negative inputs that prevents the dying ReLU problem.

Mish

Mish (Eq. (12)) function was inspired by Swish, but the Sigmoid has been replaced with a composition of Tanh and Softplus functions by Misra (2020). Mish outperformed Swish, despite similar features, in well-known benchmark tests.

$$y_{mish}(x) = x\, y_{tanh}(y_{softp}(x)) = x\, \frac{1 - e^{-\ln(1+e^x)}}{1 + e^{-\ln(1+e^x)}} \tag{12}$$

Mish (Fig. 6) is also a non-monotonic and self-regularized function. In addition to the many advantages, its complexity makes it computationally very expensive.

ELU

Exponential Linear Unit (ELU) (Eq. (13)) is a classic alternative for ReLU, where (Clevert, Unterthiner, & Hochreiter, 2016) replaced the

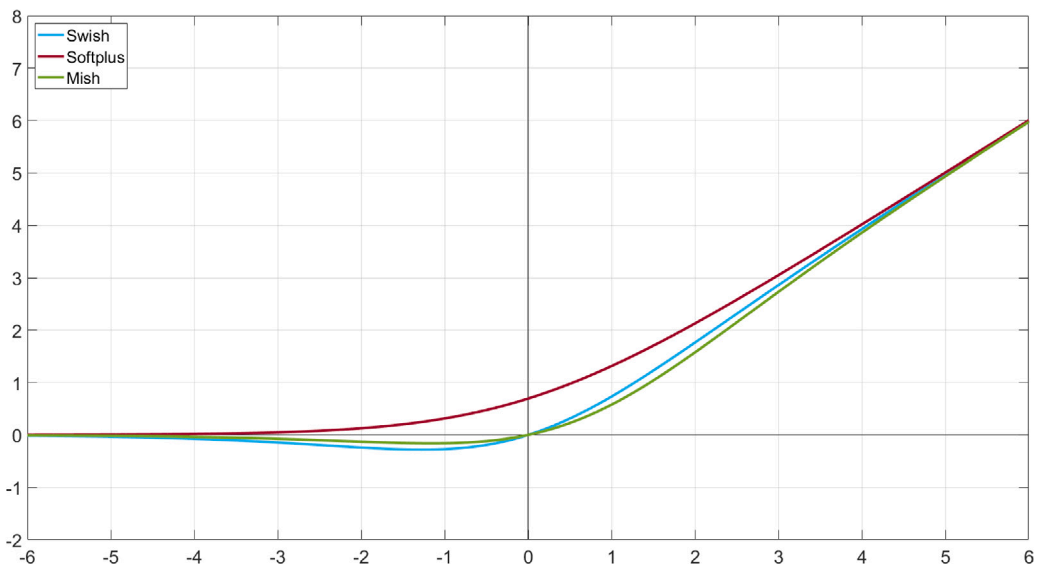


Fig. 6. Shapes of Softplus, Swish and Mish activation functions.

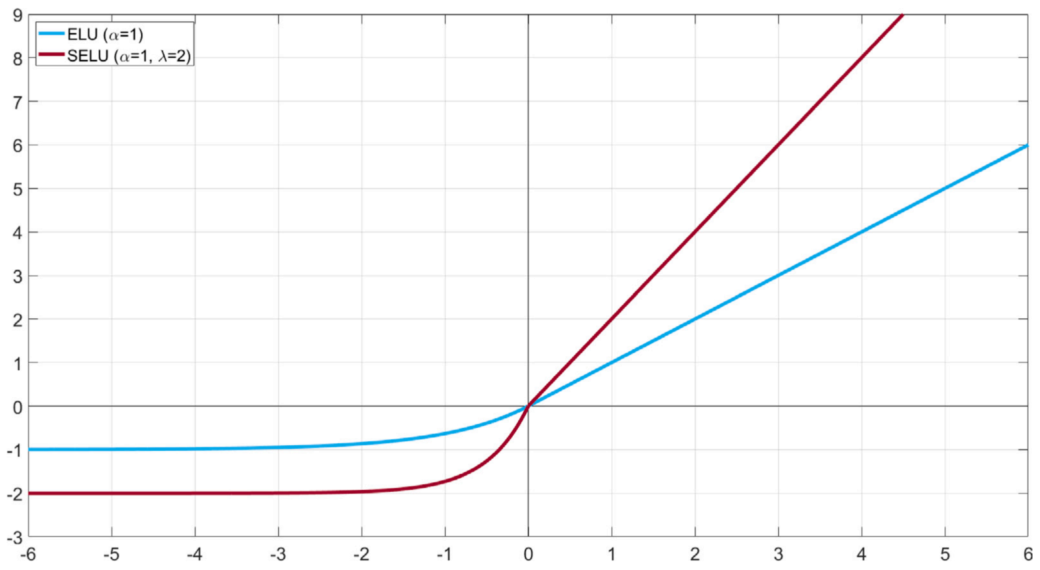


Fig. 7. Shapes of Softplus, Swish and Mish activation functions.

output of the negative part by an exponential function to eliminate the dying ReLU problem. As for Leaky ReLU and Parametric ReLU, the output can be negative, but because of the exponential term, it saturates in the negative quadrant. Disadvantages of ELU are the costly calculation and the good setting of the α parameter and the exploding gradient can also be a problem.

$$y_{elu}(x) = \begin{cases} \alpha\,(e^x - 1), & \forall x < 0 \\ x \quad , & \forall x \geq 0 \end{cases} \tag{13}$$

where $\alpha > 0$ (typically lies in the range $[0.1, 0.3]$).

SELU

Klambauer, Unterthiner, Mayr, and Hochreiter (2017) developed the Scaled Exponential Linear Unit (SELU) function (Eq. (14)). SELU is a relatively new function, which differs from ELU because it has a lambda scaling factor (Fig. 7).

$$y_{selu}(x) = \lambda \begin{cases} \alpha\,(e^x - 1), & \forall x < 0 \\ x \quad , & \forall x \geq 0 \end{cases} \tag{14}$$

Softmax

As Agarwal, Balasubramanian, and Jawahar (2018) and Martins and Astudillo (2016) confirmed, Softmax is one of the most widely used activation function for classification tasks. The Softmax function (Eq. (15)) is frequently used in the output layer. The output lies in the range [0,1] and adds up to 1. Therefore, it can handle multiple classes and perform a probability distribution consisting of n probabilities proportional to the exponentials of the input.

$$y_{softm}(x)_i = \frac{e^{x_i}}{\sum_{i=1}^{n} e^{x_i}} \tag{15}$$

where n is the number of inputs.

From the discussion above, it is evident that the development of activation functions is still ongoing. The main reason is that they also need to adapt to different new architectures and application areas according to Duan, Yang, and Dai (2022).

5. Related work

Evaluating the proper operation of neural networks was previously the subject of many research studies. The identification of a general and in-depth assessment framework of neural networks would enable the developers to identify the input variables' intervals where the investigated neural network can be reliably applied. Yu, Duan, and Ye (2022) investigated the neuron coverage as an outstandingly important evaluation factor in the case of NNs. The method aims to increase the number of evaluated neurons to identify more nested relationships between the network components and the output variables. The authors developed an advanced white-box testing method using the concept of neuron coverage that makes it possible to determine the inconsistent operation of NNs. The model identifies inactive components and continuously stimulates them to optimize neuron coverage.

Kolman and Margaliot (2005) describe a Mamdani-type all-permutations fuzzy rule base (APFRB) method. They introduced the relationship between NNs and the developed fuzzy rule base, including knowledge extraction from and knowledge insertion into NNs. Another important application possibility of the introduced model is the assessment of the dynamic behavior of different training methods.

Amjad, Liu, and Geiger (2021) analyzed the use of different indicators to interpret and evaluate fully connected feedforward NNs. Their study focused on the entropy, the mutual information with the class variable, and a class selectivity measure based on Kullback–Leibler divergence. This paper examines the relationship between the indicators introduced and decision accuracy. According to their results, mutual information and class selectivity had a positive correlation with decision accuracy, especially in the case ReLU based networks. Hayashi, Setiono, and Azcarraga (2016) introduced comprehensible classification rules based on the pruned network by evaluating the activations of the hidden neurons and the weights of the edges in the pruned network. The results of their work demonstrate that the rules derived from the neural networks could efficiently represent the classification rules. Csiszár, Csiszár, and Dombi (2020) reduced the black-box nature of NNs by integrating neural systems with continuous models and multicriteria-decision-making systems. They present that nilpotent logical systems can be applied for hybridization of the investigated models. The authors constructed the network by applying logical operators and multicriteria decision tools in the hidden layers. In their article, the focus is on the structure of integrated models and the identification of the components applicable in deep NNs.

Katz et al. (2017) introduced an efficient solver for interpreting deep NNs based on the simplex method, improved to be capable of integrating non-convex Rectified Linear Unit (ReLU) activation functions. The interpretation procedure does not apply any simplifying assumptions. The publication investigates the method on a sample network developed for airborne collision avoidance applications. The outcomes demonstrated that the developed approach can efficiently interpret network characteristics.

Tollner et al. (2022) introduced a method, where a network is trained with ReLU activation function, but replaced afterwards with a modified ReLU (*mReLU*). Because *mReLU* is continuous differentiable, a nonlinear equation system can be used to formulate analytic guarantees about the neural network's output in a given interval.

Probably, one of the most fundamental studies on neural network verification written by Pulina and Tacchella (2010) aims to represent the neural network as an abstraction and to separate safe/unsafe domains using a solver based on the Satisfiability modulo theories. The applicability of the method is critically affected by the adequacy of the abstraction. The size of the network used by Pulina and Tacchella (2011) is small/medium, as the number of hidden neurons in the different examples ranges from 5 to 20. Pulina and Tacchella (2012) demonstrated the NeVer framework through the application of the Puma 500 robot manipulator.

The Planet system introduced by Ehlers (2017b) aims to assist the Satisfiability modulo theories (SMT) solver with linear approximation based node phase assignment. The efficiency of the method is strongly determined by the approximation. The size of the network presented in the demonstration is medium, since the number of hidden neurons is 40. The case study illustrating the proposal is presented on a collision avoidance system.

Xiang, Tran, and Johnson (2018) investigates this domain as well. They aim to estimate the sensitivity of the neural network by discretizing the input domain through solving a chain of optimization problems. The network used in the demonstration is small, since the number of hidden neurons is 5. A case study of a robot arm was used to illustrate the proposal.

Bunel et al. (2020) and Katz, Barrett, Dill, Julian, and Kochenderfer (2022) focused on the verification of NNs that contain a Piecewise Linear activation function. The networks used in the demonstration are intermediate or large, as the number of hidden neurons is above 30, but some examples approach or exceed 1000. Beyond other demonstrative examples, the case studies presenting the proposal illustrate the applicability of the methods through the ACAS data set.

The comparison of the introduced NN verification methods summarized in Table 1.

6. Methodology

6.1. Verification of binary classifying neural networks

In the previous work of Tollner et al. (2022), a method for verification of binary classifying feed-forward neural networks was developed. This section briefly summarizes the previous approach to provide the basis for this publication. The basic approach is that a trained neural network can be verified, if for a certain range of values of our input variable $\vec{x} = (x_1, \dots x_\mathrm{B})$ it can be guaranteed that the network predicts only the expected class y_target in this interval:

$$\{\vec{x} \mid \forall\ x_b : l_b \le x_b \le u_b\}, b \in (1, \dots, \mathrm{B}) \implies F(\vec{x}) = y_\mathrm{target} \tag{16}$$

The set of inputs $\vec{x}$ that satisfy the boundary conditions is named $\vec{x}_I$. A feed forward network for binary classification is assumed to be structured as follows: a set of H hidden layers, each with N_h neurons ($h \in \{1, \dots, \mathrm{H}\}$), is fully connected with each previous layer, but no other layers or itself. The input layer is a set of N_0 neurons with identity activation, forming the input of $N_0 = \mathrm{B}$ features. For binary classification, the last layer of the network is assumed to consist of a single neuron equipped with a Sigmoid activation function. For this purpose, the two classes are represented by the values "0" and "1", because the Sigmoid function can map its input values to the value range of $[0, 1] \in \mathbb{R}$. Typically, a value <0.5 is then assigned to class "0", and ≥ 0.5 to class "1". In sum, the total number of neurons N in the network is

$$N = \sum_{h=0}^{\mathrm{H}} N_h, \tag{17}$$

with $N_\mathrm{H} = 1$. Aiming at the verification of NNs of this structure analytically, the problem is formulated as an equation system. Generally speaking, the function of a neuron $n_{h,i}$ can be described using two variables $x_{h,i}$ and $y_{h,i}$, being the input and output, respectively. h indicates the layer index, while i is the neuron index in the respective layer h. The input $x_{h,i}$ is calculated according to the weights of its connected predecessors. The output $y_{h,i}$ is calculated using the activation function f_h, assuming that each layer h has a common, continuously differentiable activation function. Accordingly, each neuron can be modeled with two equations; one linear equation based on the weights, and one possibly nonlinear for the activation:

$$x_{h,i} = \sum_{n=1}^{N_{h-1}} w_{y_{h-1,n}, x_{h,i}} \cdot y_{h-1,n} \tag{18}$$

Table 1
Comparison of NN verification methods.

	NeVer	Planet	SafetyVeri	BaBSR	Reluplex
Method	SMT solver	SMT with approximation	Maximum sensitivity	MILP with B&B	Modified simplex
Case study	Unimate PUMA 500 industrial manipulator	Road vehicle collision avoidance system, 3000 tuples	Robotic arm model	Numerous dataset (e.g., ACAS, PCAMNIST, etc.)	ACAS
Example network size	Small-medium	Medium	Small	Large	Medium
Applied NN representation	Abstraction	Linear approximation	Equation system	Equation system	Equation system
Demonstrated activation function	Sigmoid	Piecewise linear functions	Any monotonic function	Piecewise linear functions	Piecewise linear functions

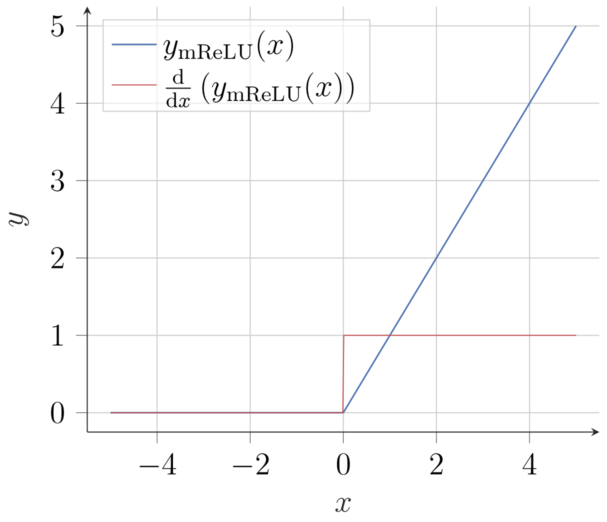


Fig. 8. The *mReLU* function and its derivative.

$$y_{h,i} = f_h(x_{h,i}) \tag{19}$$

For verification, the requirements for our input data as of Eq. (16) are formulated as boundary conditions on the system of equations. Typically, one requirement $l_b \le x_b \le u_b$ is formulated as two equations. For nonlinear activation functions (e.g. ReLU, Sigmoid, Swish), a nonlinear solver is needed. An additional requirement, that the method places on the neural network: the activation must be a closed function so that solvers for continuous problems can handle it. Therefore, to apply the approach to the commonly used ReLU function, the *mReLU* function was introduced in the previous work (see Fig. 8), which is a closed-form approximation of the ReLU function:

$$y_{\mathrm{mReLU}}(x) = \frac{\sqrt{x^2} + x}{2} \tag{20}$$

While the formulation of ReLU and *mReLU* differ, the behavior of the functions is the same. For verification purposes, this is sufficient because a neural network trained with ReLU will give the same results if *mReLU* is used instead — the derivative is only important for the training process.

Finally, the constrained optimization problem is formulated as minimizing or maximizing the output of the “last” neuron, i.e.

$$\min_{\vec{x}_t} y_{\mathrm{H},1}. \tag{21}$$

If the minimum and maximum both indicate the same class, i.e. in both are <0.5 or ≥0.5, then the nonlinear equation system (NLES) is feasible in the specified range of Eq. (16) and it can be checked whether the predicted class corresponds to the desired one. Otherwise, all that can be said about the result of the Neural Network F in the given interval is that both classes can be predicted. However, this is also an important outcome: in intervals, where different outcomes are possible, the Neural Network cannot be relied on. In summary, the workflow of the method is as follows:

1. Training of the NN.
2. If necessary, approximation of nonlinear activation functions by closed functions, which have the same behavior, but may have different derivatives.
3. Specification of intervals l_b, u_b for feature values x_b, yielding the set of target input vectors $\vec{x}_t$ and associated expected output y_{target}.
4. Formulation of the NN as a nonlinear system of equations, where two equations are formulated for each neuron as :
   (a) Based on the weights of the connections with its predecessors.
   (b) Based on the closed-form representation of the activation to obtain the output.
5. Minimization and maximization of the NLES, where the specified intervals form the constraints of the NLES.
6. Evaluation of the feasibility of the NLES based on minimum and maximum in terms of the decision boundary between the classes and the fulfillment of the specification.

6.2. Expanding on multi-class problems

For the extension to the multi-class approach, it is important to consider how a neural network is structured for this purpose. Instead of classifying the input data into two classes “0” and “1”, the network maps to C classes. In the literature, a Softmax activation function is typically used as the last layer in the network for this purpose. The Softmax layer consists of as many neurons as there are classes to predict, i.e. $N_{\mathrm{H}} = \mathrm{C}$ instead of $N_{\mathrm{H}} = 1$. Since this layer does not map to a *single* target variable $y_{\mathrm{H},1} = y_{\mathrm{target}}$ for which a minimization or maximization can be performed, a change in the method is necessary. Alternatively, instead of using the Softmax function, another final layer can be used, that maps to a single variable $y_{\mathrm{H},1}$ to which the previous optimization approach can be applied. In the following, a proposal for both possibilities is presented:

- an extension of our approach for networks with the typical Softmax activation function
- a Sigmoid-based multi-class classification, for which our approach needs only minimal modification

6.3. Verification of softmax-based multi-class networks

If the established approach to multi-class classification via the Softmax function is chosen, the output of each neuron

$$n_{\mathrm{H},i}, i \in (1, \dots, N_{\mathrm{H}}), N_{\mathrm{H}} = \mathrm{C} \tag{22}$$

in the last layer corresponds to the probability that the classified input vector $\vec{x}$ belongs to the respective class i. The predicted class c is yielded by the index of the neuron $n_{\mathrm{H},c}$ whose output $y_{\mathrm{H},c}$ is the maximum of the outputs $y_{\mathrm{H},i}, i \in \{1, \dots, \mathrm{C}\}$, i.e.,

$$c = \arg\max_{i \in \{1,\dots,C\}} y_{\mathrm{H},i}. \tag{23}$$

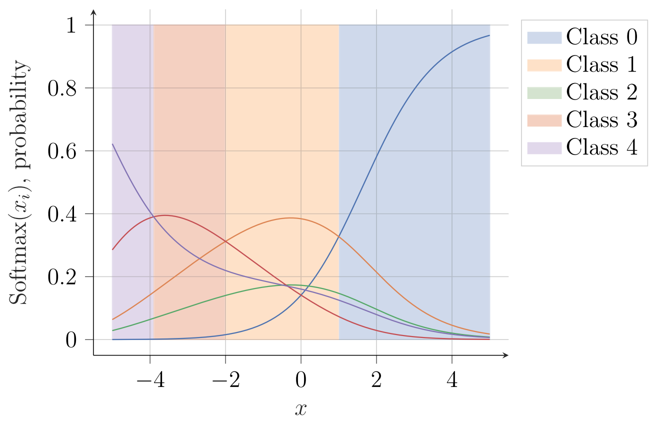


Fig. 9. Exemplary Softmax-based multi-class classification.

Thinking of the neural network F whose output is in $[0,1]^{\mathrm{C}}$ (corresponding to the output of the classifying neurons $y_{\mathrm{H},1}$ to $y_{\mathrm{H,C}}$), the neural network can be verified for a certain interval and desired target class $y_{\mathrm{target}} = c, c \in \{1,\dots,\mathrm{C}\}$. Verification is successful if the minimum of the output of the neuron corresponding to the desired target class $y_{\mathrm{H},c}$ is larger than all maxima of the other neurons, i.e.

$$\min_{\vec{x}_t} y_{\mathrm{H},c} \geq \max_{\vec{x}_t} y_{\mathrm{H},i}, \forall\ i \in \{1,\dots,\mathrm{C}\} \tag{24}$$

Figuratively speaking, the activation functions of the output layer $f_i\ \forall i \in \{1,\dots,\mathrm{C}\}$ and f_c is above all other functions f_i. An example of the multi-class classification is given in Fig. 9, where the Softmax function is applied to artificial data of five classes. In Fig. 9, the functions represents the class probabilities, the color of the background indicates the dominant class with the largest probability related to the investigated interval. Since, we assume that we have a priori knowledge about the proper operation of the system, with other words about the expected target class related to given input values, we can evaluate if the system behaves according to our expectations. For example, when the output should be "0" for $2 < x < 4$ based on the system specification, then the verification is successful, because the probability of class "0" is larger than all the other class probabilities in this interval. To implement the verification of Softmax-based multi-class neural networks in a common solver, one can take advantage of a property of the Softmax layer. As the Softmax mapping is strictly monotonic function, it does not change the ratios between the neurons of the layer, i.e., the neuron that provides the minimum of the output values of all neurons of the Softmax layer for a given input $\vec{x}$ also has the minimum input value. Accordingly it can be formulated as

$$\arg\min_{i\in\{1,\dots,\mathrm{C}\}} y_{\mathrm{H},i} = \arg\min_{i\in\{1,\dots,\mathrm{C}\}} x_{\mathrm{H},i}, \tag{25}$$

and $\arg\max_{i\in\{1,\dots,\mathrm{C}\}}$ respectively. Therefore, for the global determination of maximum and minimum, the Softmax function can be neglected. Instead, the network is considered only up to $x_{\mathrm{H},i}$ and C NLES are minimized and maximized respectively for $x_{\mathrm{H},c}$. In addition to verifying a guaranteed prediction of the class y_{target}, an alternative use case can be pursued, the guaranteed non-prediction of certain classes y_{ex}. For this case, the reverse is checked to see if there is always a class above the class y_{ex} to be excluded:

$$\{f_{\mathrm{i}} | \max_{\vec{x}_t} f_{\mathrm{ex}} < \min_{\vec{x}_t} f_i\}, f_i, f_{\mathrm{ex}} \in \mathcal{F} \implies f_{\mathrm{ex}} \notin \mathcal{F}_C, \mathcal{F}_C \subset \mathcal{F} \tag{26}$$

In summary, the following procedure emerges as the modified algorithm for the verification of multi-class feed-forward NNs:

1. Training the NN with Softmax Output and cross-entropy loss.
2. If necessary, approximation of nonlinear activation functions by closed functions, which have the same behavior, but may have different derivatives.
3. Specification of intervals l_b, u_b for feature values x_b, yielding the set of target input vectors $\vec{x}_t$ and associated expected output y_{target} (2 and 3 same as in the original version).
4. Formulation of the NN as a nonlinear system of equations, where two equations are formulated for each neuron of layers 1 to $\mathrm{H}-1$ as:
   - Based on the weights of the connections with its predecessors.
   - Based on the closed-form representation of the activation to obtain the output.
5. For each of the classes $c \in \{1,\dots,\mathrm{C}\}$:
   - Form a sub-system of equations, extending the equation system of (4) with only the equations containing the weights of $x_{\mathrm{H},c}$, connection $n_{\mathrm{H},c}$ to the previous layer $\mathrm{H}-1$.
   - Solve the NLES for the minimum and maximum in x_t.
6. Evaluate feasibility and verification based on minimum of NLES for target class y_{target} in comparison to maxima of other classes.

6.4. Verification of sigmoid-based multi-class networks

Instead of solving the Softmax-based approach using multiple optimization problems, it is also possible to represent a multi-class problem with a Sigmoid-based network. A network of this structure has only one neuron in the last layer, and can therefore be minimized or maximized using the previous approach. To do this, the Sigmoid function must be able to predict C values instead of only distinguishing between two classes 0 and 1. To this end, in particular, the decision for a class is adjusted by dividing the Sigmoid function more finely. Depending on the number of classes, the output of the Sigmoid function is divided into equally sized sections. For example, for five classes at $0 \leq y \leq 0.2$ class "0" is predicted, $0.2 < y \leq 0.4$ class "1", etc. (see Fig. 10(b)). With four classes, the boundaries change accordingly to 0.25, 0.5, and 0.75 respectively (see Fig. 10(a)). Thus, the Sigmoid function serves primarily to limit the output values, while being a continuous and closed-form representative function. While the system of equations does not change, the training process needs some modification. According to the number of classes, the decision boundaries are calculated and the target values of the classes are set for training. Target values for training are the mean value between two decision boundaries. For example with four classes, the target values are 0.125 for class "0", 0.375 for class "1", etc. Instead of using a loss metric for classification, training is done with a regression metric. Finally, the feasibility evaluation must be modified for verification. A neural network can then be verified for a given interval of input data if the minimum and maximum of the output are within the decision bounds of the Sigmoid function for the desired class. The only difference to binary classification is that in general two decision boundaries have to be checked instead of only against 0.5.

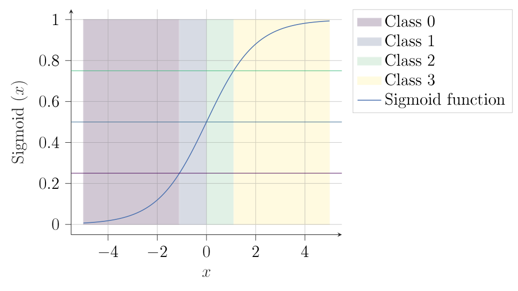
7. Results

In the following, we present the results using three examples of multi-class classification: a self-generated toy dataset, the Iris dataset, and a self-generated use case from the automotive sector. For the toy dataset and the Iris dataset, we only discuss the results on the Softmax classification.

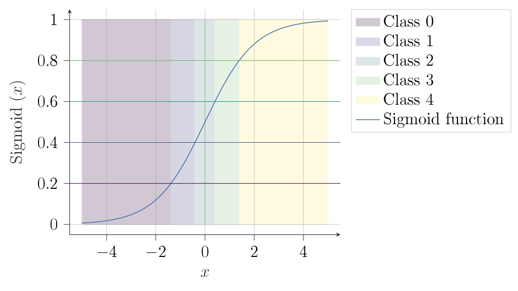
7.1. Toy dataset

As a first problem, we generated 10,000 samples with three classes ("1", "2", "3") and two features (x_1 and x_2) using `make_blobs` from the *sklearn* library. In addition, the data points were multiplied by a matrix A:

$$\mathrm{A} = \begin{bmatrix} 0.6 & -0.6 \\ -0.4 & 0.8 \end{bmatrix} \tag{27}$$



(a) Four-class classification



(b) Five-class classification

Fig. 10. Sigmoid-based multi-class classification.

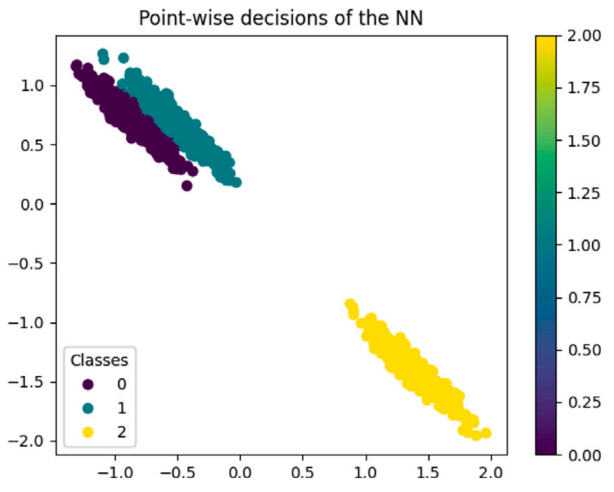


Fig. 11. Point-wise predictions for the toy data test set.

Table 2
Numerical verification results for the toy dataset.

Approach	Class	Minimum	Maximum
Softmax	0	−5.9105	−2.3626
	1	−0.9536	1.3732
	2	1.4339	2.1518

Of the total amount of samples, we used 20% as unseen test data and 16% as validation data for early stopping and tracking the generalization in the training process. We standardized the data by estimating the mean and variance from the training dataset and using this to scale the training, validation and test datasets. On the normalized data, we trained a network with one input layer, two hidden layers with 20 neurons each, and one output layer. In the following, we only discuss the results with Swish activation and Softmax classification. Fig. 11 shows the point-wise classification results on the test data set. The neural net successfully identified the three generated clusters, with an accuracy of 99.85% on the test dataset. Our method shall now be used to validate that new data points that are similar to the training data of class “2” are correctly classified. As an example, we therefore want to show that an area around the data points of the yellow cluster in Fig. 11 is completely assigned to class “2”.

For this problem, we define the following specification interval for the two features x_1 and x_2:

$$1.0 \leq x_1 \quad \leq 1.5 \tag{28}$$

$$-1.5 \leq x_2 \quad \leq -1.0 \tag{29}$$

Our assumption is that within this interval, which only contained data points from class “2” in the training, only class “2” is predicted. The specification interval defined by the Eqs. (28) and (29) can be validated, as the results in Table 2 show. As assumed, class “2” dominates the prediction of the neural network in this area: The minimum before the Softmax function for class “2” is higher than the maxima of classes “0” and “1”. This means that class “2” is always predicted based on the Softmax function if a data point lies within the specification interval of the Eqs. (28) and (29).

7.2. Iris dataset

As a second example, we consider the Iris dataset. This small dataset consisting of 150 samples classifies three species of the iris plant (setosa, virginica, versicolor) based on four features. The four features are the *petal_length*, *petal_width*, *sepal_length* and *sepal_width*, which describe the flower shape and size of the plant. Since only 150 samples are available, we chose a smaller test set of 10% of the data points and 18% for validation. The network architecture in this example is very similar to the one we used for the toy dataset: an input layer, two hidden layers with 20 neurons each, and an output layer. We used min–max normalization instead of standardization here, as this provided slightly better results on the validation dataset. The network thus achieves an accuracy of 93.3% on the test dataset (cf. Fig. 12(b)) with Swish activation function and Softmax classification. Based on the training data (Fig. 12(a)), we can see that the “setosa” class can be separated from the other two classes using the *petal_length* feature. We now want to validate if the network has learned this knowledge from the data. For data that is similar to the “setosa” data from the training, “setosa” should therefore be predicted. We express this as a specification interval as follows:

$$4.5 \leq sepal\_length \quad \leq 5.0 \tag{30}$$

$$2.0 \leq sepal\_width \quad \leq 4.5 \tag{31}$$

$$1.0 \leq petal\_length \quad \leq 2.0 \tag{32}$$

$$0.0 \leq petal\_width \quad \leq 0.5 \tag{33}$$

The numerical outputs of our tool are shown in Table 3. From the inf and $>10^5$ values, we conclude that the solver does not find a minimum and maximum for the three classes. Accordingly, the neural network cannot be verified. It can therefore not be ruled out that a class other than “setosa” is predicted for certain inputs in the specified interval. While this may look like an undesirable result, it is a very important conclusion, helping to develop safer NNs. Even though the accuracy on the test set seems to be viable (>90%), the Iris dataset may not contain enough samples to reliably train a NN with safe generalization performance, or the test dataset may be too small to assess the true generalization performance.

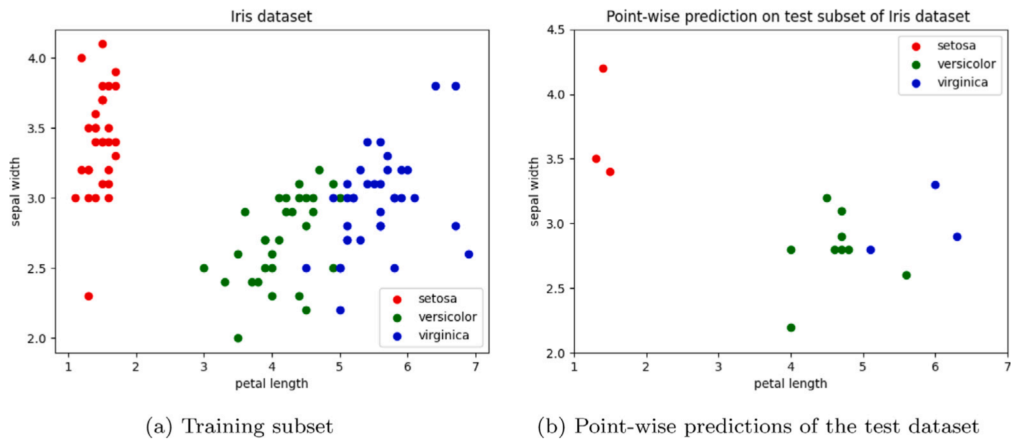


Fig. 12. Results on the Iris dataset.

Table 3
Numerical verification results for the Iris dataset.

Approach	Class	Minimum	Maximum
	Setosa	– inf	inf
Softmax	Versicolor	– inf	inf
	Virginica	$-5 \cdot 10^5$	$5 \cdot 10^5$

7.3. Use case: emergency braking assist

As an example, we illustrate the two proposed approaches using the "emergency braking assistant" use case from the automotive sector. Emergency braking assistants will be mandatory in new vehicles in the future. In the present work, a neural network is designed to classify the action to be selected based on two features (x_1 and x_2), the distance d to a standing object in front of the ego vehicle, and the speed v_0 of the ego vehicle.

7.3.1. Data generation

The five classes for which data is generated describe, "do nothing", "warning", "preload brakes", "braking", and "crash/preload airbag". The feature space is uniformly sampled to generate training data. The required stopping distance is calculated based on the stopping sight distance, assuming 1.5 s reaction time of the driver,

$$s_{\text{ssd}}(v_0) = 1.5v_0 + \frac{v_0^2}{2 \cdot \mu_{\text{H}}\, \text{g}} = 1.5v_0 + \frac{v_0^2}{2a}, \tag{34}$$

with g being the gravitational acceleration and μ_{H} the effective friction coefficient. The friction coefficient depends on tires and street conditions, with typical values for dry streets ≥ 0.8. For the classification problem, the maximum available deceleration $\mu_{\text{H}} \cdot \text{g}$ with $\mu_{\text{H}} = 0.8$ is scaled from 25%, 50%, 75% to 100% yielding the four discrete deceleration values $a \in \{a_{25}, \dots, a_{100}\}$. The maximum deceleration is thus around $7.85\,\text{m}\,\text{s}^{-2}$ for the class "braking". The required distance to stop, depending on speed and μ_{H} is shown in Fig. 13. Comparing the required stopping distance when applying the four different discrete decelerations to the available distance, the label l for training is generated as the minimum deceleration sufficient to stop within the available distance,

$$\begin{aligned} l(v_0, d) &= \min a, \\ &\quad s.t.\ \{a \in \{a_{25}, \dots, a_{100}\} \mid d - s_{\text{ssd}}(v_0, a) > 0\} \end{aligned} \tag{35}$$

The fifth class "Crash" is assigned if none of the possible braking forces is sufficient to come to a stop in front of the object. Using this procedure, we generated a data set of 100.000 samples which was split randomly into 70% training data and 30% test data. The training data set was randomly undersampled to yield a balanced distribution of classes. In addition, the data was standardized to have zero mean and unit variance. An additional validation data set was not used.

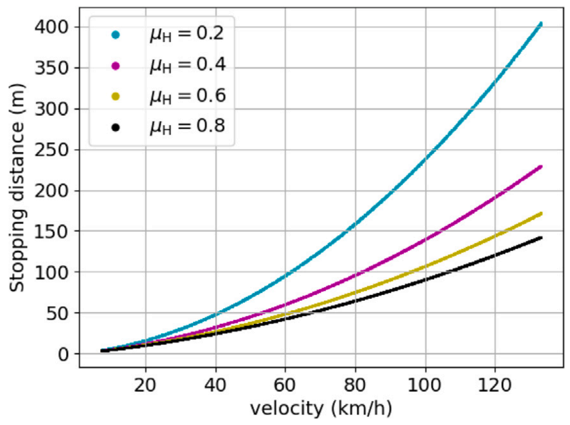


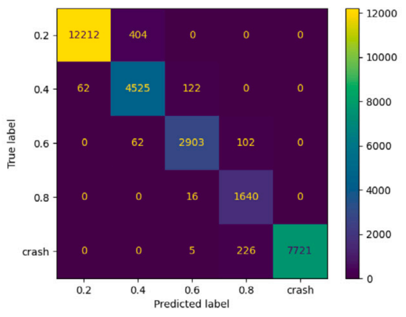
Fig. 13. Required distance to stop for different values of μ_{H}.

7.3.2. Neural networks training

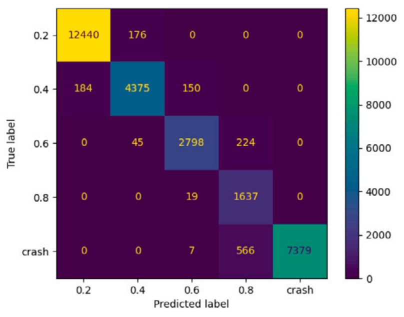
For each of the two variants of our method, a neural network is trained. The main difference is in the last layer, where Softmax activation is used for method A and Sigmoid activation for method B. Since the Softmax layer has significantly more connections than the Sigmoid layer with only one neuron, for the same number of layers the amount of trainable parameters is lower for Sigmoid activation. To compensate for this, the Sigmoid-based approach (Fig. A.16(b)) uses a network with four hidden layers and the Softmax approach (Fig. A.16(a)) uses two hidden layers. Implemented with Keras and Tensorflow 2.1, the networks have 53 (Softmax) and 61 (Sigmoid) trainable parameters. In the hidden layers, an activation that can be represented in a closed form must be used for the applicability of the verification method described here. In this work, the previously introduced *mReLU* function is compared with the Swish activation. The Softmax network is trained with categorical crossentropy loss function, the Sigmoid network with mean squared error loss, since technically a regression on five target values is trained. Figs. 14(a), 14(b) and Table 4 show the training process of the two types of networks. Both converge quickly, showing a slightly superior performance of the Swish-based networks compared to the ReLU networks. The confusion matrices in Figs. 14(a) and 14(b) show the class-wise results, yielding comparable performance of the Softmax and the Sigmoid based approaches.

7.3.3. Verification

For demonstration of the verification approach, two exemplary specification intervals are defined. In the first case, a specific class is



(a) Confusion matrix of the Softmax-based net



(b) Confusion matrix of the Sigmoid-based net

Fig. 14. Training results.

Table 4
Training results for the emergency braking use case.

Approach	Softmax		Sigmoid	
Activation	ReLU	Swish	ReLU	Swish
Accuracy (Train)	0.959	0.968	0.903	0.951
Accuracy (Test)	0.969	0.967	0.920	0.954

Table 5
Numerical verification results for case 1.

Approach	Class	Minimum	Maximum
Softmax	0	49.3254	81.7770
	1	38.8690	58.3571
	2	29.5243	45.8864
	3	−77.3205	−49.5316
	4	−286.3853	−182.6587
Sigmoid		0.1186	0.1449

Table 6
Numerical verification results for case 1, refined intervals.

Approach	Intervals ($50 \leq x_1 \leq 60$)	Class	Minimum	Maximum
Softmax	$150 \leq x_2 \leq 160$	0.2	49.3254	63.1035
		0.4	38.8690	44.9161
Softmax	$160 \leq x_2 \leq 170$	0.2	54.2292	67.8348
		0.4	42.2932	48.3080
Softmax	$170 \leq x_2 \leq 200$	0.2	59.0736	81.7770
		0.4	45.7041	58.3571

expected to be the result. The expected outcome, given the specification intervals

$$50 \leq x_1 \leq 60 \tag{36}$$

$$150 \leq x_2 \leq 200 \tag{37}$$

is for the class "do nothing", i.e. a deceleration of 25% or around $1.96\,\mathrm{m\,s^{-2}}$ would be sufficient to come to a stop before the leading object. The feature x_1 represents the speed of the ego vehicle, and x_2 the distance to the leading object. The minimum and maximum outputs of the Sigmoid network for the given intervals are 0.1186 and 0.1449, respectively (see bottom row of Table 5). The predicted class is 0 for the given intervals as both values fall between the decision boundaries of negative infinity and 0.2. This confirms the validity of the Sigmoid-based classification approach for these intervals, as the expected and predicted classes are the same. The upper rows of Table 5 provide the minimum and maximum before calculating the Softmax function to provide a better link to the verification intervals.

It can be observed, that class "0"'s minimum is larger than the maxima of classes 2, 3, and 4. Class "1"'s maximum is larger than class "0"'s minimum. Thus, class "1" can not be excluded. However, one can look at refined verification intervals. Analyzing the results for

$$150 \leq x_2 \leq 160, \tag{38}$$

$$160 \leq x_2 \leq 170, \text{ and} \tag{39}$$

$$170 \leq x_2 \leq 200 \tag{40}$$

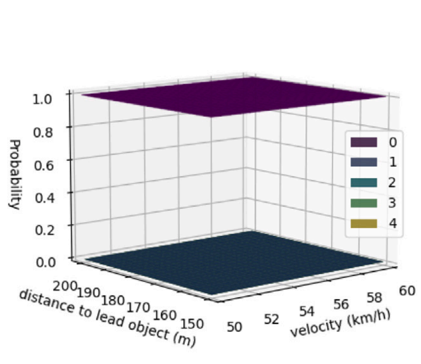
separately, all intervals can be verified to result in class "0" (see Table 6). As such, the Softmax-based network will predict class "1" for all other data points in the given intervals. Graphically, verification results are shown in Fig. 15. As expected from the precedent interpretation of Table 5, Fig. 15(a) shows class "0" predicted across the whole interval presented in (36) and (37). The other classes are all clustered in the lower part of the graph and are therefore not visually distinguishable. One of the major challenges of the method is also evident from the graph. After applying the Softmax function, class "0" clearly dominates graphically. Verification cannot be performed for the entire interval in one step due to strongly varying values over the whole interval. On larger specification intervals, a specific class is generally above the other classes. However, its minimum can still be below the maximum of another class. Therefore, if the values are widely spread out before applying the Softmax function, a smaller verification interval is necessary. The Sigmoid based approach in Fig. 15(b) yields a similar interpretation. For further clarification, the sector division of Fig. 10 as well as the projection of the results on the upper part of the graph is added. As in the Softmax case, the Sigmoid classification can be clearly specified to class "0", as the range of values lies within the specified limits.

Looking at a second case,

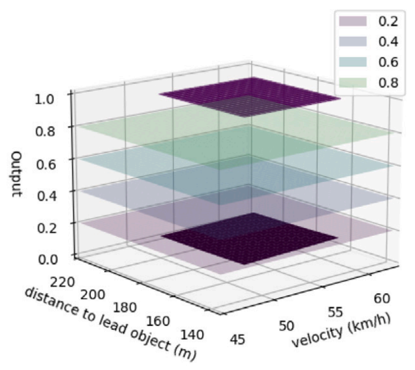
$$60 \leq x_1 \leq 80 \tag{41}$$

$$20 \leq x_2 \leq 60 \tag{42}$$

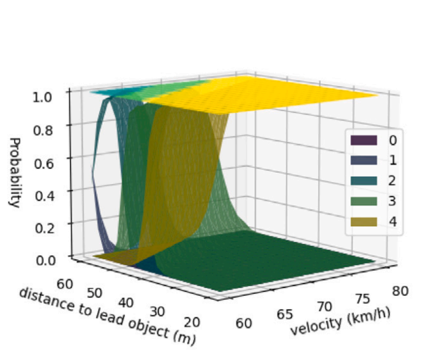
we would expect that the network does predict classes "1", "2", "3" or "4", because the ego speed (x_1) is higher but the distance to the lead object (x_2) is lower. Thus, a braking initiation would be expected. The results in Table 7 indicate, that for the Sigmoid network, this can be verified. Class "0" would require that the outputs minimum is below 0.2, which is not the case. The same conclusion can be drawn in Fig. 15(d). In contrast, for the Softmax network, the verification is not possible, because the maximum of class "0" (−2.1213) is larger than the minima of all the other classes. Thus, class "0" cannot be excluded from the possible outputs. It is worth mentioning, that in the corresponding Fig. 15(c) class "0" is barely visible but is a possible outcome within the specified boundaries. This shows the superiority of using a formal method instead of graphical approaches. In both surface graphs 15(c) and 15(d), the transition between the classes over the specification interval are identifiable. Fig. 15(c) shows the typical distribution of the Softmax function which reinforces the differentiation between the dominant class and the other class.



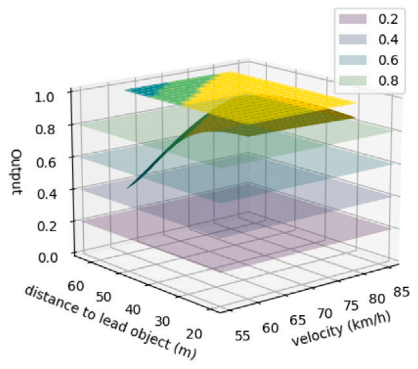
(a) Case 1, Softmax



(b) Case 1, Sigmoid



(c) Case 2, Softmax



(d) Case 2, Sigmoid

Fig. 15. Verification results for both verification intervals. For the Softmax network, z-axis shows the probability of the predicted class. For the Sigmoid network plots, the z-axis shows the output of the Sigmoid function with the classification boundaries.

Table 7
Numerical verification results for case 2.

Approach	Class	Minimum	Maximum
Softmax	0	−36.8660	−2.1213
	1	−28.4942	4.8156
	2	−6.9890	4.5517
	3	−7.1433	12.4059
	4	−28.7765	25.3166
Sigmoid		0.3893	0.9275

8. Conclusion

In this work, a verification method for multi-class feed-forward neural networks was developed. The method extends the previous work of Tollner et al. (2022) by providing a formalization of the verification principle required for the multiclass setting. With the new method, multi-class feed-forward NNs using either a standard Softmax classification, or a Sigmoid-based classification, can be verified. A nonlinear solver is used to find minimum and maximum of the network output in the given specification intervals of the input features. The approach was successfully demonstrated on an example related to the automotive industry with artificial data. However, a challenge, especially for the Softmax-based networks, is the specification of an appropriate observation interval.

For future work, we plan to extend the approach to other types of data, e.g. image data. However, our primary assumption, that the network neurons only contain activation functions that have a continuous closed-form representation limits the use of off-the-shelf techniques for image processing, e.g. Convolutional NNs. In addition, methodological extensions to specify and refine the verification areas could be developed. Moreover, the approach will be evaluated on real-world data and more sophisticated NNs, where verification becomes even more crucial.

CRediT authorship contribution statement

Daniel Grimm: Conceptualization, Methodology, Writing – original draft, Data curation. **Dávid Tollner:** Conceptualization, Methodology, Writing. **David Kraus:** Methodology, Writing – original draft. **Árpád Török:** Methodology, Writing, Supervision, Investigation. **Eric Sax:** Review & editing. **Zsolt Szalay:** Review & editing.

Declaration of competing interest

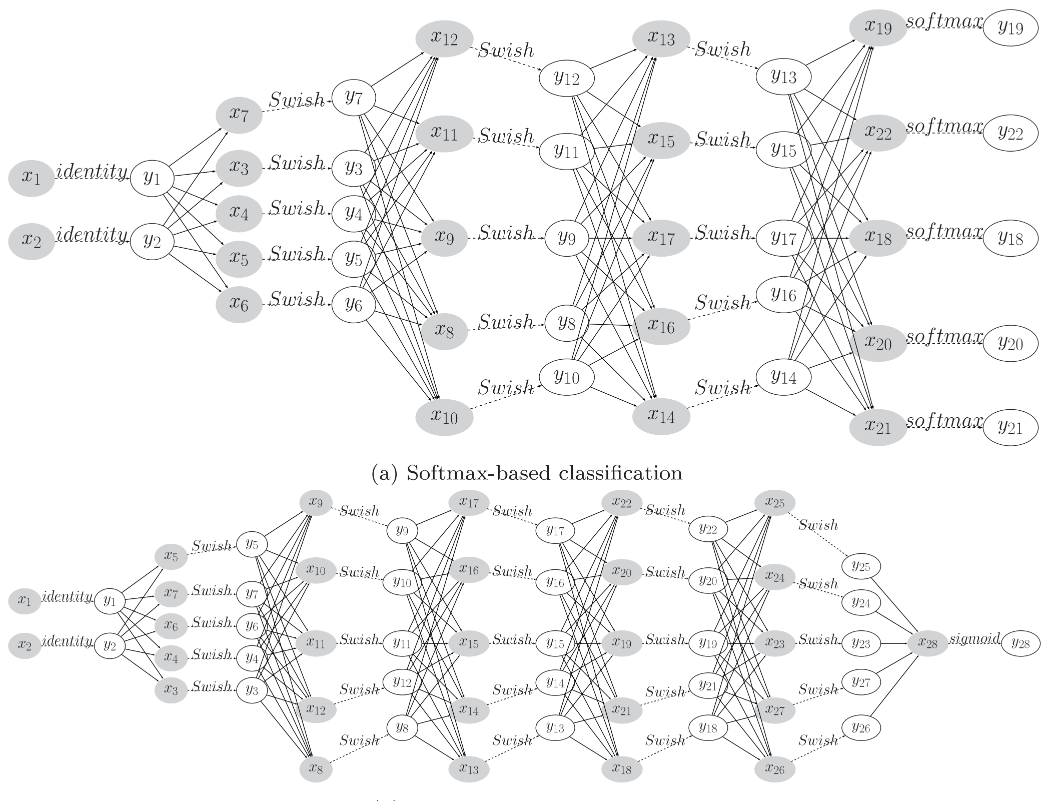
The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

Prepared with the professional support of the Doctoral Student Scholarship Program of the Co-operative Doctoral Program of the Ministry for Innovation and Technology from the source of the National Research, Development and Innovation Fund. The research was supported by the Ministry of Innovation and Technology NRDI Office within the framework of the Autonomous Systems National Laboratory Program.



(a) Softmax-based classification

(b) Sigmoid-based classification

Fig. A.16. Structure of the used multi-class neural networks.

Appendix. Neural networks structure

See Fig. A.16.

References

Agarwal, N., Balasubramanian, V. N., & Jawahar, C. (2018). Improving multiclass classification by deep networks using DAGSVM and triplet loss. *Pattern Recognition Letters*, *112*, 184–190.

Amari, S.-i. (1998). Natural gradient works efficiently in learning. *Neural Computation*, *10*(2), 251—276.

Amjad, R. A., Liu, K., & Geiger, B. C. (2021). Understanding neural networks and individual neuron importance via information-ordered cumulative ablation. *IEEE Transactions on Neural Networks and Learning Systems*.

Barrachina, D. G.-L., Boldizsar, A., Zoldy, M., & Torok, A. (2019). Can neural network solve everything? Case study of contradiction in logistic processes with neural network optimisation. In *2019 modern safety technologies in transportation* (pp. 21–24). IEEE.

Brown, R. A., Schmerling, E., Azizan, N., & Pavone, M. (2022). A unified view of SDP-based neural network verification through completely positive programming. In *International conference on artificial intelligence and statistics* (pp. 9334–9355). PMLR.

Bunel, R., Lu, J., Turkaslan, I., Torr, P. H., Kohli, P., & Kumar, M. P. (2020). Branch and bound for piecewise linear neural network verification. *Journal of Machine Learning Research*, *21*(42), 1–39.

Bunel, R. R., Turkaslan, I., Torr, P., Kohli, P., & Mudigonda, P. K. (2018). A unified view of piecewise linear neural network verification. *Advances in Neural Information Processing Systems*, *31*.

Cao, H., & Zoldy, M. (2021). MPC tracking controller parameters impacts in roundabouts. *Mathematics*, *9*(12), 1394.

Clevert, D.-A., Unterthiner, T., & Hochreiter, S. (2016). Fast and accurate deep network learning by exponential linear units (ELUs). arXiv:1511.07289.

Csiszár, O., Csiszár, G., & Dombi, J. (2020). Interpretable neural networks based on continuous-valued logic and multicriteria decision operators. *Knowledge-Based Systems*, *199*, Article 105972.

Duan, B., Yang, Y., & Dai, X. (2022). Feature activation through first power linear unit with sign. *Electronics*, *11*(13), http://dx.doi.org/10.3390/electronics11131980.

Dugas, C., Bengio, Y., Bélisle, F., Nadeau, C., & Garcia, R. (2000). Incorporating second-order functional knowledge for better option pricing. In T. Leen, T. Dietterich, & V. Tresp (Eds.), *Vol. 13*, *Advances in neural information processing systems*. MIT Press.

Ehlers, R. (2017a). Formal verification of piece-wise linear feed-forward neural networks. In D. D'Souza, & K. Narayan Kumar (Eds.), *Automated technology for verification and analysis* (pp. 269–286). Cham: Springer International Publishing.

Ehlers, R. (2017b). Formal verification of piece-wise linear feed-forward neural networks. In *Automated technology for verification and analysis: 15th international symposium, ATVA 2017, Pune, India, october 3–6, 2017, proceedings 15* (pp. 269–286). Springer.

Hayashi, Y., Setiono, R., & Azcarraga, A. (2016). Neural network training and rule extraction with augmented discretized input. *Neurocomputing*, *207*, 610–622.

Hornik, K., Stinchcombe, M., & White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, *2*(5), 359—366.

Katz, G., Barrett, C., Dill, D. L., Julian, K., & Kochenderfer, M. J. (2017). Reluplex: An efficient SMT solver for verifying deep neural networks. In *International conference on computer aided verification* (pp. 97–117). Springer.

Katz, G., Barrett, C., Dill, D. L., Julian, K., & Kochenderfer, M. J. (2022). Reluplex: a calculus for reasoning about deep neural networks. *Formal Methods in System Design*, *60*(1), 87–116.

Klambauer, G., Unterthiner, T., Mayr, A., & Hochreiter, S. (2017). Self-normalizing neural networks. *Advances in Neural Information Processing Systems*, *30*.

Kolman, E., & Margaliot, M. (2005). Are artificial neural networks white boxes? *IEEE Transactions on Neural Networks*, *16*(4), 844–852.

Luenberger, D. G., Ye, Y., et al. (1984). *Vol. 2*, *Linear and nonlinear programming*. Springer.

Maas, A. L., Hannun, A. Y., & Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. *Vol. 30*, In *Proceedings of the 30th international conference on machine learning*.

Martins, A., & Astudillo, R. (2016). From softmax to sparsemax: A sparse model of attention and multi-label classification. In *International conference on machine learning* (pp. 1614–1623). PMLR.

Mekonnen, A. A., & Sipos, T. (2022). Crash prediction models and methodological issues. *Periodica Polytechnica Transportation Engineering*, *50*(3), 267–272.

Misra, D. (2020). Mish: A self regularized non-monotonic activation function. arXiv:1908.08681.

Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted Boltzmann machines. *Vol. 27*, In *Proceedings of the 27th international conference on international conference on machine learning* (pp. 807–814). Omni Press.

Pulina, L., & Tacchella, A. (2010). An abstraction-refinement approach to verification of artificial neural networks. In *Computer aided verification: 22nd international conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. proceedings 22* (pp. 243–257). Springer.

Pulina, L., & Tacchella, A. (2011). N e v er: a tool for artificial neural networks verification. *Annals of Mathematics and Artificial Intelligence*, *62*, 403–425.

Pulina, L., & Tacchella, A. (2012). Challenging SMT solvers to verify neural networks. *Ai Communications*, *25*(2), 117–135.

Ramachandran, P., Zoph, B., & Le, Q. V. (2017). Searching for activation functions. http://dx.doi.org/10.48550/ARXIV.1710.05941, URL https://arxiv.org/abs/1710.05941.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, *65*(6), 386––408.

Struye, J., & Latré, S. (2020). Hierarchical temporal memory and recurrent neural networks for time series prediction: An empirical validation and reduction to multilayer perceptrons. *Neurocomputing*, *396*, 291–301.

Tollner, D., Ziyu, W., Zöldy, M., & Török, Á. (2022). Demonstrating a new evaluation method on ReLU based neural networks for classification problems. *TBD*, *X*(X), X.

Vishnukumar, H. J., Butting, B., Müller, C., & Sax, E. (2017). Machine learning and deep neural network — Artificial intelligence core for lab and real-world test and validation for ADAS and autonomous vehicles: AI for efficient and quality test and validation. In *2017 intelligent systems conference (intelliSys)* (pp. 714–721). http://dx.doi.org/10.1109/IntelliSys.2017.8324372.

Vu, H. T., & Lim, J. (2022). Effects of country and individual factors on public acceptance of artificial intelligence and robotics technologies: a multilevel SEM analysis of 28-country survey data. *Behaviour & Information Technology*, *41*(7), 1515–1528. http://dx.doi.org/10.1080/0144929X.2021.1884288.

Xiang, W., Tran, H.-D., & Johnson, T. T. (2018). Output reachable set estimation and verification for multilayer neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, *29*(11), 5777–5783.

Yu, J., Duan, S., & Ye, X. (2022). A white-box testing for deep neural networks based on neuron coverage. *IEEE Transactions on Neural Networks and Learning Systems*.