Neural Networks 138 (2021) 14–32

Contents lists available at ScienceDirect

Neural Networks

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

Review

A survey on modern trainable activation functions

Andrea Apicella [a,*], Francesco Donnarumma [b], Francesco Isgrò [a], Roberto Prevete [a]

[a] Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione, Università di Napoli Federico II, Italy
[b] Institute of Cognitive Sciences and Technologies (ISTC), National Research Council (CNR), Via San Martino della Battaglia 44, 00185 Rome, Italy

ARTICLE INFO

Article history:
Received 17 May 2020
Received in revised form 17 December 2020
Accepted 25 January 2021
Available online 9 February 2021

Keywords:
Neural networks
Machine learning
Activation functions
Trainable activation functions
Learnable activation functions

ABSTRACT

In neural networks literature, there is a strong interest in identifying and defining activation functions which can improve neural network performance. In recent years there has been a renovated interest in the scientific community in investigating activation functions which can be trained during the learning process, usually referred to as *trainable*, *learnable* or *adaptable* activation functions. They appear to lead to better network performance. Diverse and heterogeneous models of trainable activation function have been proposed in the literature. In this paper, we present a survey of these models. Starting from a discussion on the use of the term "activation function" in literature, we propose a taxonomy of trainable activation functions, highlight common and distinctive proprieties of recent and past models, and discuss main advantages and limitations of this type of approach. We show that many of the proposed approaches are equivalent to adding neuron layers which use fixed (**non-trainable**) activation functions and some simple local rule that constrains the corresponding weight layers.

Contents

1. Introduction

The introduction of new activation functions has contributed to renewing the interest of the scientific community in neural networks, having a central role for the expressiveness of artificial neural networks. For example, the use of ReLU (Glorot, Bordes, & Bengio, 2011), Leaky ReLU (Maas, Hannun, & Ng, 2013), parametric ReLU (He, Zhang, Ren, & Sun, 2015), and similar activation functions (for example, Clevert, Unterthiner, & Hochreiter, 2016; Dugas, Bengio, Bélisle, Nadeau, & Garcia, 2000), has shown to improve the network performances significantly, thanks to properties such as the no saturation feature that helps to avoid typical learning problems as vanishing gradient (Bishop, 2006). Thus, individuating new activation functions that can potentially improve the results is still an open field of research. In this research line, an investigated and promising approach is the possibility to determine an *appropriate* activation function by learning. In other words, the key idea is to involve the activation functions in the learning process together (or separately) with the other parameters of the network such as weights and biases, thus obtaining a *trained activation function*. In literature

* Corresponding author.
E-mail address: andrea.apicella@unina.it (A. Apicella).

https://doi.org/10.1016/j.neunet.2021.01.026
0893-6080/

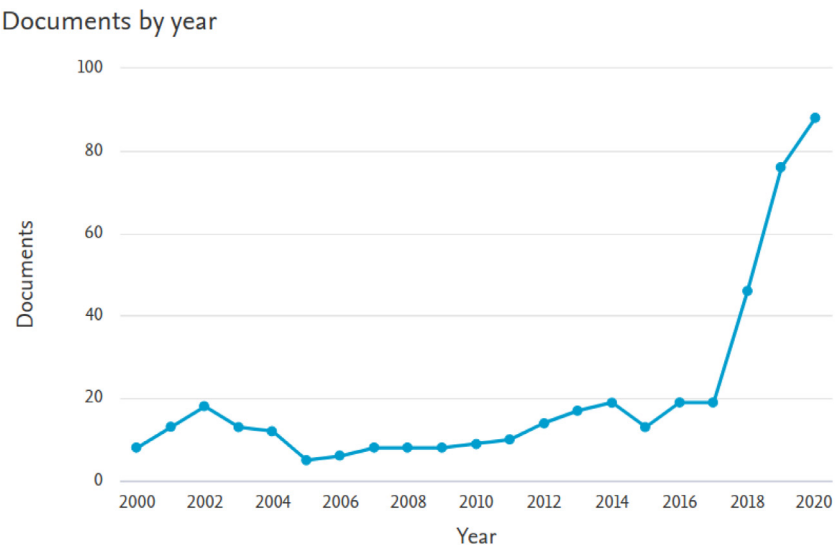

Fig. 1. Number of papers by year on trainable activation functions .
Source: Scopus, query: (((((“trainable activation function” OR “learnable activation function”) OR “adaptive activation function”) OR “adaptable activation function”) AND “neural networks”)).

we usually find the expression “trainable activation functions”, however the expressions “learneable”, “adaptive” or “adaptable” activation functions are also used, see, for example, Apicella, Isgrò, and Prevete (2019), Qian, Liu, Liu, Wu, and Wong (2018) and Scardapane, Van Vaerenbergh, Totaro, and Uncini (2019). Many and heterogeneous trainable activation function models have been proposed in literature, and in recent years there has been a particular interest in this topic, see Fig. 1.

In this paper, we present a survey of trainable activation functions in the neural network domain, highlighting general and peculiar proprieties of recent and past approaches. In particular, we examine trainable activation functions in the context of feed-forward neural networks, although many of the approaches that we are going to discuss can also be applied to recurrent neural networks. In the first place, the relevant and critical properties of these approaches are isolated. Taking into consideration this analysis, we propose a taxonomy that characterizes and classifies these functions according to their definition. Moreover, we show that many of the proposed approaches are equivalent to adding neuron layers which use fixed activation functions (**non-trainable** activation functions) and some simple local rule that constrains the corresponding weight layers. Also, based on this taxonomy, we discuss the expressivity of these functions and the performances that were achieved when neural networks with trainable activation functions were used.

In order to explain and better analyze the various models of trainable activation function, we start describing what is usually meant by the expressions “feed-forward neural network” and “activation function”, summarizing the main non-trainable (fixed) activation functions proposed in literature so far.

Definitions and symbols

Multi-Layer Feed Forward (MLFF) networks are composed of N elementary computing units (neurons), which are organized in $L > 1$ layers. The first layer of an MLFF network is composed of d input variables. Each neuron i belonging to a layer l, with $1 \leq l \leq L$, may receive connections from all the neurons (or input variables in case of $l = 1$) of the previous layer $l - 1$. Each connection is associated with a real value called *weight*.

The flow of computation proceeds from the first layer to the last layer (*forward propagation*). The last neuron layer is called *output* layer, the remaining neuron layers are called *hidden* layers. The computation of a neuron i belonging to the layer l corresponds to a two-step process: the first step is the computation of the neuron input a_i^l is computed; the second step is the computation of the neuron output z_i^l. The neuron input a_i^l is usually constructed as a linear combination of its incoming input values, corresponding to the output of the previous layer: $a_i^l = \sum_j w_{ij}^l z_j^{l-1} + b_i^l$ where w_{ij}^l is the weight of the connection going to the neuron j belonging to the layer $l - 1$ to the neuron i belonging to the layer l, b_i^l is a parameter said *bias*, z_j^{l-1} is the output of the neuron j belonging to the layer $l - 1$ (or the input variables, if $l = 1$), and j runs on the indexes of the neurons of the layer $l - 1$ (or the input variables, if $l = 1$) which send connections to the neuron i. In a standard matrix notation, a_i^l can be expressed as $a_i^l = \mathbf{w}_i^{lT} \mathbf{z}^{l-1} + b_i$, where $\mathbf{w}_i^l$ is the weight column vector associated with the neuron i belonging to the layer l and $\mathbf{z}^{l-1}$ is the column vector corresponding to the output of the neurons belonging to the layer $l - 1$. If $l = 1$ the vector $\mathbf{z}^{l-1}$ corresponds to the input variables. The neuron output z_i^l is usually computed by a differentiable, non linear function $f(\cdot)$: $z_i^l = f(a_i^l)$. In this network model, the nonlinear functions $f(\cdot)$ are generally chosen as simple *fixed* functions such as the *logistic sigmoid* or the *tanh* functions, and they are usually called *activation functions*.

Activation functions: a brief historical excursus

The expression *activation function* has not always been used with today’s meaning, since other expressions have been used in literature, as *transfer function* or *output function*. In some cases, *transfer function* is used as synonym of *activation function* (such as in Hagan, Demuth, & Beale, 1996) but, in other cases, there is a clear distinction between the different forms. For instance, in a well known survey (Duch & Jankowski, 1999), the two forms “activation function” and “output function”, assume different meanings; more precisely: “the activation function determines the total signal a neuron receives. The value of the activation function is usually scalar and the arguments are vectors. The second function determining neuron’s signal processing is the output function [...], operating on scalar activations and returning scalar values. Typically a squashing function is used to keep the output values within specified bounds. These two functions together determine the values of the neuron outgoing signals. The

composition of the activation and the output function is called the transfer function". In a nutshell, Duch and Jankowski (1999) distinguish among:

- *activation function* $I(\mathbf{z})$: an internal transformation of the input values $\mathbf{z}$. The most common artificial neuron model makes a weighted sum of the input values, that is $I(\mathbf{z}) = \mathbf{w}^T\mathbf{z}+b$ where $\mathbf{w}$, b are called neuron parameters (or weights) and the bias, respectively. However, in the recent literature the statement *activation function* loses this meaning, as described in the following;
- *output function* $o(a)$: a function which returns the output value of the neuron using the activation value $a = I(\mathbf{z})$, i.e. $o : a \in \mathbb{R} \rightarrow o(a) \in \mathbb{R}$. However, the largest part of recent literature usually refers to this function as *activation function*, so giving a different meaning respect to the word *activation*;
- *transfer function* $T(\mathbf{z})$: the composition of output function and activation function, that is $T(\mathbf{z}) = o(I(\mathbf{z}))$.

This distinction between activation, output and transfer function was not used in previous research works, such as in Haykin (1994), where an activation function is "[...] a squashing function in that it squashes (limits) the permissible amplitude range of the output signal to some finite value". Here, it is clear that the "activation function" is what Duch and Jankowski (1999) define as "output function". The terminology and the formalization used in Haykin (1994) are the most used in literature. For example, DasGupta and Schnitger (1993) define an activation function as a member of "a class of real-valued functions, where each function is defined on some subset of $\mathbb{R}$".

In Goodfellow, Bengio, and Courville (2016) an activation function is "a fixed nonlinear function". The nonlinearity requirement comes from Cybenko (1989) and Hornik, Stinchcombe, and White (1989) where it is shown that the activation functions have to be non-constant, bounded and monotonically-increasing continuous to ensure the network's universal approximator property (see Section 3). In Müller, Reinhardt, and Strickland (2012) the activation function is introduced as "A transfer function f_i [...] defined for each [network] node i, which determines the state of the node as a function composed of its bias, the weights of incoming links, and the states of nodes connected to it", so using again the term *transfer* and *activation* in an interchangeable manner. In Eldan and Shamir (2016) an activation function is clearly defined as any $f : \mathbb{R} \rightarrow \mathbb{R}$ function. All these definitions agree in defining an activation function as a functional mapping between two subsets of the real numbers, provided that this function meets suitable requirements to guarantee the MLFF network's universal approximator property.

An exception seems to be Hagan et al. (1996), where the expressions "activation function" and "transfer function" are used indistinctly as synonyms of what the authors in Duch and Jankowski (1999) call "output function". More precisely, in Hagan et al. (1996) the authors define the activation/transfer function as the function which "produces the scalar neuron output [...]. The transfer function [...] may be a linear or a nonlinear function [...]. A particular transfer function is chosen to satisfy some specification of the problem that the neuron is attempting to solve".

In another direction, part of literature about trainable activation functions loses the concept of activation function as reported in works such as DasGupta and Schnitger (1993), Goodfellow et al. (2016) and Haykin (1994), proposing instead new neuron architectures which seem to work without a clearly-defined activation function, generating an output that is different from a simple non-linearity applied to a linear combination of the neuron inputs. These approaches seem to change all the internal neuron behavior, so, instead of trainable activation functions, if we want to keep the distinction reported in Duch and Jankowski (1999), we should refer to these as trainable transfer functions.

Summarizing, the behavior of an artificial neural network (ANN) is typically characterized by the elementary computations of each neuron (Bishop, 2006; Bishop et al., 1995; Leshno, Lin, Pinkus, & Schocken, 1993; Njikam & Zhao, 2016), and the elementary computational process of each neuron is generally a two-step process (Bishop, 2006; Bishop et al., 1995). In the first one, a real value is computed from the neuron input values and the incoming connection weights. Usually, this value is computed as a weighted linear combination of the neuron's input, the standard inner-product in $\mathbb{R}^d$ between the vectors $\mathbf{x}$ and $\mathbf{w}$. Different definitions have been proposed in the literature (see, for example, Chen, Cowan, & Grant, 1991). In the second step, a scalar output value is computed by a Real activation function, $f : \mathbb{R} \longrightarrow \mathbb{R}$. From a historical point of view, different definitions of activation functions can be found in the literature as we discussed previously in this section. However, the current literature still identifies two distinct steps (Bishop, 2006; Bishop et al., 1995; Scardapane et al., 2019) in the computation at the neuron level. Therefore, in this paper, in order to highlight this two-step process, unless otherwise specified, we distinguish among (1) an input function $I : \mathbb{R}^d \longrightarrow \mathbb{R}$ characterizing the first step of the neuron computation, (2) an activation function $f : \mathbb{R} \longrightarrow \mathbb{R}$ which characterizes the second step, and (3) a transfer function $T : \mathbb{R}^d \longrightarrow \mathbb{R}$ as the composition of I and f, $T = f(I(z))$, characterizing the whole neuron computation. In this way, a neuron output depends directly on the incoming input values (input values of the whole network or outputs of other neurons) and connection weights only. Furthermore, in the context of non-fixed activation functions the words *trainable*, *learnable* and *adaptable* are used as synonyms.

In this work, we give three main contributions; the first one is a survey on the current state of the art of trainable activation functions and the obtained results. The second one consists of highlighting relevant and critical properties of these approaches. Taking into consideration this analysis, we propose a taxonomy that characterizes and classifies these functions according to their definition. The last contribution is to show that, in many cases, using a trainable activation is equivalent to using a deeper neural network model with additional constraints on the parameters.

The work is so organized as follows: in Section 2 we propose a possible activation function taxonomy, distinguishing them between fixed and trainable ones. In Section 3 we give a brief summary of the most used fixed activation functions, while in Sections 4 and 5 we make a survey of the state of art of the trainable activation functions; in Section 6 we discuss the obtained performances in literature. Section 7 is left to final remarks.

2. A taxonomy of activation functions

In this work, we propose a possible taxonomy of the main activation functions presented in literature, see Fig. 2. As stated in the previous section, we focus on activation functions as defined in Goodfellow et al. (2016) and Haykin (1994), i.e., the output of a neuron is computed by a two-step process: first the input of the neuron is computed by a functional mapping from $\mathbb{R}^d$ to $\mathbb{R}$ (usually a weighted sum), then the output (or activation) of the neuron is computed by the activation function which is a functional mapping from $\mathbb{R}$ to $\mathbb{R}$. This way to compute the neuron output is widely used in literature and de facto it is the standard in artificial neural networks. However, a number of significant neural network models which implement different approaches have been presented in literature. Among these models we have isolated a subset of them which can be interpreted as "trainable non standard neuron definitions". In our taxonomy we put them

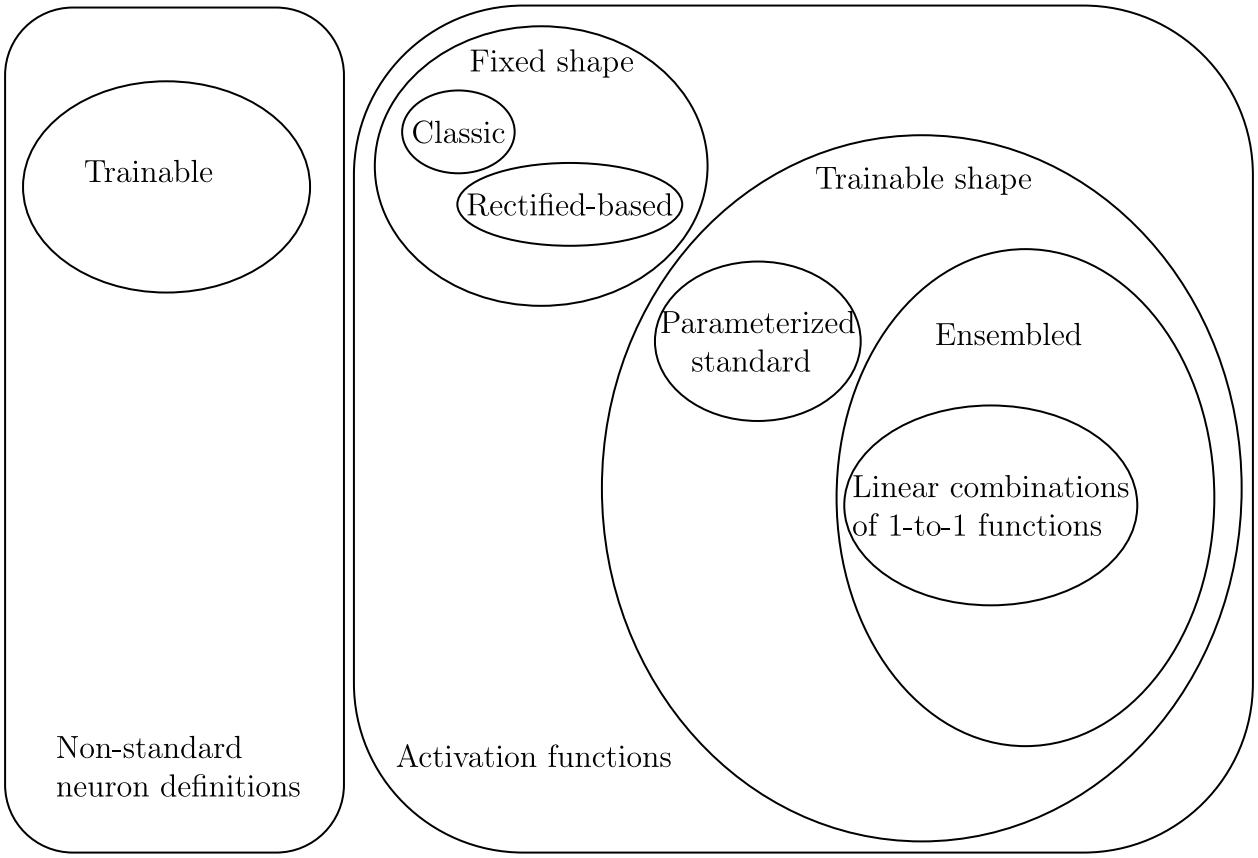

Fig. 2. A proposed taxonomy of the activation functions proposed in Neural Networks literature.

as a distinct class, and we discuss trainable non standard neuron definitions in Section 5.

The primary classification is based on the possibility of modifying the activation function shape during the training phase. So, one can isolate two main categories:

- **fixed-shape activation functions**: all the activation functions with a fixed shape, for example, all the classic activation functions used in neural network literature, such as sigmoid, tanh, ReLU, fall into this category. However, since the introduction of rectified functions (as ReLU) can be considered a turning point in literature contributing to improving the neural network performances significantly and increasing the interest of the scientific community, we can further divide this class of functions into:
  - rectified-based function: all the functions belonging to the rectifier family, such as ReLU, LReLU, etc.
  - classic activation function: all the functions that are not in the rectifier family, such as the sigmoid, tanh, step functions.
- **trainable activation functions**: this class contains all the activation functions the shape of which is learned during the training phase. The idea behind this kind of functions is to search a good function shape using knowledge given by the training data. However, we will show that several trainable activation functions can be reduced to classical feed-forward neural subnetworks composed of neurons equipped with classical fixed activation functions, grafted into the main neural network only by adding further layers. In other terms, a neural network architecture equipped with trainable activation functions can have a similar (if not the same) behavior of a deeper network architecture equipped with just classic (fixed) activation functions, in some cases by adding simple constraints on the network parameters, e.g., by fixing some weights or sharing them (as in convolutional networks) and by proper arranging of the layers.

  Taking into account all these considerations, among all the trainable activation functions described into the literature we can isolate two different families:
  - *parameterized standard functions*: in this case, we consider all the trainable functions derived from standard fixed activation functions with the addition of a set of trainable parameters. In other words, they are defined as a parameterized version of a standard fixed function whose parameter values are learned from data. We will see later that several of these functions can be expressed in terms of subnetworks.
  - *Functions based on ensemble methods*: they are defined by mixing several distinct functions. A common way to mix different functions is combining them linearly, i.e., the final activation functions are modeled in terms of linear combinations of one-variable functions. We group together all these activation functions in a subclass that we named *linear combination of one-to-one functions*. In this case, these one-variable functions can, in turn, have additional parameters. Many of these approaches can be expressed in terms of MLFF subnetworks which receive just one single input value, as we will discuss in . By contrast, some activation functions are already proposed, in the original papers, in terms of one or more sub-networks which can be in turn described as a linear combination of one-to-one functions. Moreover, other functions are modeled in an

Table 1
Some of the most used fixed activation functions.

Name	Expression	Range
Identity	$\mathrm{id}(a) = a$	$(-\infty, +\infty)$
Step (Heaviside)	$Th_{\geq 0}(a) = \begin{cases} 0 & \text{if } a < 0 \\ 1 & \text{otherwise} \end{cases}$	$\{0, 1\}$
Bipolar	$B(a) = \begin{cases} -1 & \text{if } a < 0 \\ +1 & \text{otherwise} \end{cases}$	$\{-1, 1\}$
Sigmoid	$\sigma(a) = \frac{1}{1+e^{-a}}$	$(0, 1)$
Bipolar sigmoid	$\sigma_B(a) = \frac{1-e^{-a}}{1+e^{-a}}$	$(-1, 1)$
Hyperbolic tangent	$\tanh(a)$	$(-1, 1)$
Hard hyperbolic tangent	$\tanh_H(a) = \max(-1, \min(1, a))$	$[-1, 1]$
Absolute value	$\mathrm{abs}(a) = \lvert a \rvert$	$[0, +\infty)$
Cosine	$\cos(a)$	$[-1, 1]$

analytic form. We will show that also in this case, many of these functions can be modeled as subnetworks nested into the main network architecture.

The next two sections describe the distinctive features of activation function classes which have been isolated by our taxonomy and discuss some advantages and limitations of the different approaches.

3. Fixed-shape activation functions

With the expression "fixed-shape activation functions", we indicate all the activation functions which are defined without parameters that can be modified during the training phase.

Since many trainable activation functions in literature are proposed as a combination or variation of fixed-activation functions, this section presents a brief description of the main fixed-shape activation functions used in neural networks. Several studies which compare different fixed activation functions have been made over the years, see for example Nwankpa, Ijomah, Gachagan, and Marshall (2018), Pedamonti (2018), Sibi, Jones, and Siddarth (2013) and Xu, Huang, and Li (2016), Xu, Wang, Chen, and Li (2015).

This section is divided into two parts. The former is dedicated to the classic activation functions used primarily in the past. The latter is about ReLU and its possible improvements given by changing its basic shape.

3.1. Classic activation functions

A short and partial list of the most common activation functions used in Neural Network literature is given in Table 1 and shown in Fig. 3.

In Cybenko (1989) and Hornik et al. (1989) it is shown that any continuous function defined on a compact subset, can be approximated arbitrarily well by a feed-forward network with a single hidden layer (*shallow network*), provided that the number of hidden neurons is sufficiently large and the activation functions are non-constant, bounded and monotonically-increasing continuous functions. This theorem demonstrated that activation functions like the identity function or any other linear function, used for example in early Neural Networks as ADALINE or MADALINE (Widrow & Hoff, 1960; Widrow & Lehr, 1990), cannot approximate any continuous function. Therefore, for many years, bounded activation functions such as sigmoid (Cybenko, 1988) or hyperbolic tangent (Chen, 1990) have been the most used activation functions for neural networks.

Over the years, several studies have shown how bounded activation functions could reach excellent results, especially in shallow network architectures (see for example Glorot et al., 2011; Pedamonti, 2018). Unfortunately, the training of networks equipped with these functions suffers from the *vanishing gradient problem* (see Bengio, Simard, & Frasconi, 1994) when networks with many layers (Deep Neural Networks, DNN) are used, compromising the network training. In Pinkus (1999) and Sonoda and Murata (2017) it was shown that the requirements specified in Cybenko (1989) and Hornik et al. (1989) for the activation functions to give to a network the universal approximation property were too strong, showing that also neural networks equipped with unbounded but non-polynomial activation functions (e.g. ReLU, Nair & Hinton, 2010) are universal approximators. Furthermore, unbounded activation functions seem to attenuate the vanishing gradient problem (see Nair & Hinton, 2010), opening new frontiers in neural networks and machine learning research.

3.2. Rectifier-based activation functions

Over the last years, many different activation functions have been proposed, most of which inspired by the success obtained by ReLU (Glorot et al., 2011), and therefore based on a similar shape, with small variations, compared to the original function. This kind of activation functions is the standard *de facto* in current neural network architectures, overcoming other such classic functions as sigmoid and tanh used in the past. One of the first studies that showed the performance improvements of networks equipped with rectifier-based activation functions was (Glorot et al., 2011), where DNNs equipped with ReLU activation functions improved the performances if compared to networks with sigmoid units. As stated above, the main benefit of using rectified activation functions is to avoid the vanishing gradient problem (Bengio et al., 1994), which has been one of the main problems for deep networks for many years. Various efforts continue to be made in the scientific community to find new activation functions to improve neural network performances. However, the research on classic activation functions has not come to an end, for example, properties on these functions are still discussed in recent studies (Gulcehre, Moczulski, Denil, & Bengio, 2016; Xu et al., 2016).

In the remainder of this subsection, we identify the main characteristics of ReLU and ReLU-family functions, some of which are shown in Fig. 4.

ReLU. This function, defined as $\mathrm{ReLU}(a) = \max(0, a)$, has significant positive features:

- It alleviates the vanishing gradient problem being not bounded in at least one direction;
- It facilitates *sparse coding* (Montalto, Tessitore, & Prevete, 2015; Tessitore & Prevete, 2011), as the percentage of neurons that are really active at the same time is usually very low. The benefits of sparsity are described in Glorot et al. (2011) and can be resumed in a better dimensionality of the representation and a more significant invariance to slight input changes.

However, ReLU function is not clear from defects, such as the "dying" ReLU problem (Maas et al., 2013) or the non-differentiability at zero, that appears when a large negative bias is learned causing the output of the neuron to be always zero regardless of the input. Consequently, in the following, we discuss some ReLU variants.

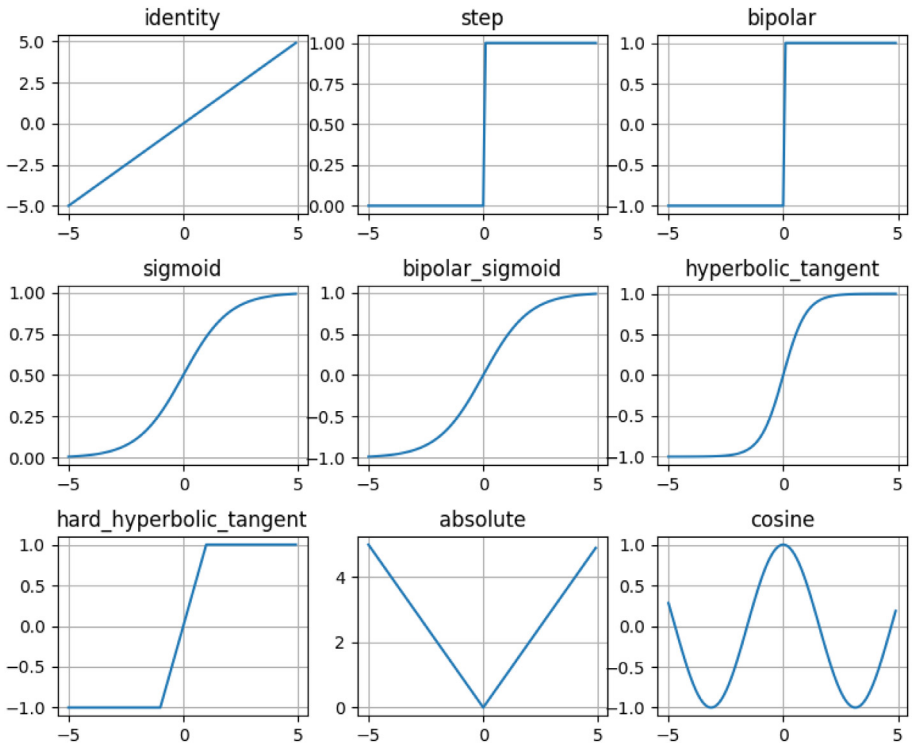

Fig. 3. Examples of classic activation functions.

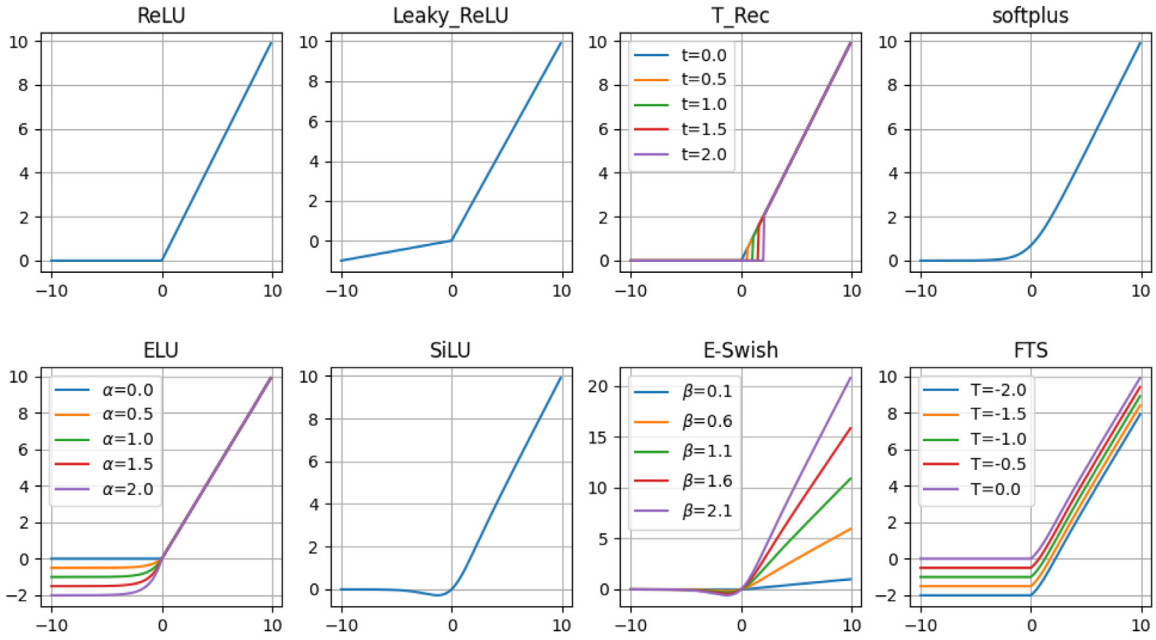

Fig. 4. Examples of rectifier-based activation functions.

Leaky ReLU. One of the earliest rectified-based activation functions based on ReLU was LReLU (Maas et al., 2013).

LReLU function was an attempt to alleviate the potential problems of the ReLU mentioned above. It is defined as:

$$\mathrm{LReLU}(a) = \begin{cases} a & \text{if } a > 0 \\ 0.01 \cdot a & \text{otherwise.} \end{cases}$$

A Leaky Rectifier Activation Function allows the unit to give a small gradient when the unit is saturated and not active, i.e., when $a \leq 0$. However, the authors show empirically that Leaky rectifiers perform nearly identically to standard rectifiers, resulting in a negligible impact on network performances.

A randomized version (Randomized Leaky ReLU), where the weight value for a is sampled by a uniform distribution $U(l, u)$ with $0 \leq l < u < 1$ was also proposed in Xu et al. (2015).

Truncated rectified. Authors in Memisevic, Konda, and Krueger (2014), tackle the problem to find alternatives to ReLU focusing on a particular type of DNN (Autoencoders), starting from the observation that this type of network tends to have large negative bias, which can have several side effects on the learning process

(see Memisevic et al., 2014 for further details). From this observation, the authors propose the Truncated Rectified as activation function which can be defined as:

$$\mathrm{TRec}_t(a) = \begin{cases} a & \text{if } a > t \\ 0 & \text{otherwise.} \end{cases}$$

Note that the t point is a non-continuity point, unlike ReLU in which the threshold point (which is 0) is only a non-differentiable point. Authors use TRec only during training, and then replace it with ReLU during testing. The authors make this choice in order to obtain both sparse coding and the minimizing of the error function without any kind of weight regularization.

Softplus. Introduced by Dugas et al. (2000), the *softplus* function can be seen as a smooth approximation of ReLU function. It is defined as

$$\mathrm{softplus}(a) = \log(1 + \exp(a))$$

The smoother form and the lack of points of non-differentiation could suggest a better behavior and an easier training as an activation function. However, experimental results (Glorot et al., 2011) tend to contradict this hypothesis, suggesting that ReLU properties can help supervised training better than softplus functions.

Exponential linear unit (ELU). Introduced in Clevert et al. (2016), the ELU is an activation function that keeps the identity for positive arguments but with non-zero values for negative ones. It is defined as:

$$\mathrm{ELU}(a) = \begin{cases} a & \text{if } a > 0 \\ \alpha \cdot (\exp(a) - 1) & \text{otherwise.} \end{cases}$$

where α controls the value for negative inputs. The values given by ELU units push the mean of the activations closer to zero, allowing a faster learning phase (as shown in Clevert et al., 2016), at the cost of an extra hyper-parameter (α) which requires to be set.

Sigmoid-weighted linear unit. Originally proposed in Elfwing, Uchibe, and Doya (2018), Sigmoid-weighted Linear Unit is a sigmoid function weighted by its input, i.e.:

$$\mathrm{SiLU}(a) = a \cdot \mathrm{sig}(a)$$

In the same study, the derivative of SiLU is also proposed as activation function, i.e.:

$$\mathrm{dSiLU}(a) = \mathrm{sig}(a)\big(1 + a(1 - \mathrm{sig}(a))\big)$$

These functions have been tested on reinforcement learning tasks. Moreover, further applications are given in Ramachandran, Zoph, and Le (2018) where the same function is tested with the name of Swish-1.

E-swish. A SiLU variation is proposed in Alcaide (2018) by adding a multiplicative coefficient to the SiLU function, and obtaining:

$$\text{E-swish}_\beta(a) = \beta \cdot a \cdot \mathrm{sig}(a)$$

with $\beta \in \mathbb{R}$. The function name comes from the Swish activation function, a trainable version of SiLU function proposed in Ramachandran et al. (2018) (see Section 4.1 for further details). However, E-Swish has no trainable parameters, leaving to the user the tuning of the β parameter. The authors consider the β parameter as a hyper-parameter to be found by a search procedure.

Flatten-T Swish. The function described in Chieng, Wahid, Pauline, and Perla (2018) has properties of both ReLU and sigmoid, combining them in a manner similar to the Swish function.

$$\mathrm{FTS}(a) = \begin{cases} a \cdot \frac{1}{1+\exp(-a)} + T, & \text{if } x \geq 0 \\ T, & \text{otherwise.} \end{cases}$$

When $T = 0$ the function becomes $\mathrm{ReLU}(a) \cdot \mathrm{sig}(a)$, a function similar to Swish-1, where the ReLU function substitutes the identity. T is an additional fixed threshold value to allow the function to return a negative value (if $T < 0$), differently from the classic ReLU function. The authors plan to propose a method to learn the parameter T in future work.

4. Trainable activation functions

The idea of using trainable activation functions is not new in the neural networks research field. Many studies were published on this subject as early as the 1990s (see, for example, Chen & Chang, 1996; Guarnieri, 1995; Piazza, Uncini, & Zenobi, 1992, 1993). In more recent years, the renewed interest in neural networks has led the research to consider again the hypothesis that trainable activation functions could improve the performance of neural networks.

In this section we describe and analyze the main methods presented in the literature related to the activation functions that can be learned by data. Relying on their main characteristics, we can isolate two distinct families:

- Parameterized standard activation functions.
- Activation functions based on ensemble methods.

With *parameterized standard activation* functions we refer to all the functions with a shape very similar to a given fixed-shape function, but tuned by a set of trainable parameters; with *ensemble methods* we refer to any technique merging different functions.

In the following of this section we discuss these two families of functions.

4.1. Parameterized standard activation functions

As mentioned above, with the expression "parameterized standard activation" functions we refer to all the functions with a shape very similar to a given fixed-shape function, but having a set of trainable parameters that let this shape to be tuned. The addition of these parameters, therefore, requires changes, even minimal ones, in the learning process; for example, when using gradient-based methods, the partial derivatives of these new parameters are needed.

In the remainder of this subsection, we focus on the main functions which fall into this family, some of which are represented in Fig. 5.

Adjustable generalized sigmoid. A first attempt to have a trainable activation function was given in Hu and Shao (1992). The proposed activation function was a generalization of the classic sigmoid $\mathrm{sig}(a) = \frac{1}{1+\exp(-a)}$ with the addition of two trainable parameters α, β to adjust the function shape, i.e.:

$$\mathrm{AGSig}(a) = \frac{\alpha}{1 + \exp(-\beta a)}$$

Both parameters are learned together using a gradient descent approach based on the backpropagation algorithm to compute the derivatives of the error function with respect to the network parameters.

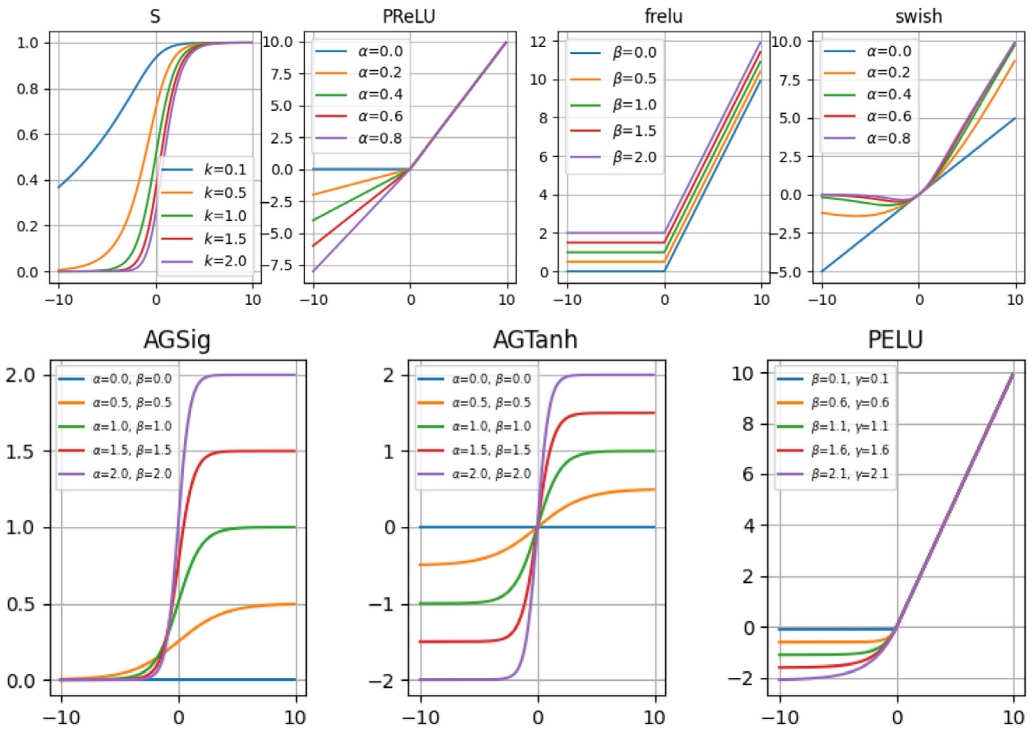

Fig. 5. Examples of parameterized standard activation functions.

Sigmoidal selector. In Singh and Chandra (2003) the following class of sigmoidal functions is proposed as:

$$S_k(a) = \left(\frac{1}{1+\exp(-a)}\right)^k$$

These functions are parameterized by a value $k \in (0, +\infty)$. In Chandra and Singh (2004) the parameter k is learned (and so the effective function is selected) together with the other network parameters by the gradient descent and backpropagation algorithms.

Adjustable generalized hyperbolic tangent. Proposed in Chen and Chang (1996), this activation function generalizes the classic hyperbolic tangent function $\tanh(a) = \frac{1-e^{-2a}}{1+e^{-2a}}$ introducing two trainable parameters α, β:

$$\text{AGTanh}(a) = \frac{\alpha\big(1-\exp(-\beta a)\big)}{1+\exp(-\beta a)}$$

In this function, α adjusts the saturation level, while β controls the slope. These two parameters are learned together with the network weights using the classic gradient descent algorithm combined with back-propagation, initializing all the values randomly. In Yamada and Yabuta (1992a, 1992b) a similar activation function was proposed, with the main difference that it included only one trainable parameter controlling the slope.

Parametric ReLU. He et al. (2015) introduced another ReLU-like function which partially learns its shape from the training set, in fact it can modify the negative part using a parameter α; the function Parametric ReLU, can be defined as:

$$\text{PReLU}(a) = \begin{cases} a & \text{if } a > 0 \\ \alpha \cdot a & \text{otherwise.} \end{cases}$$

The additional parameter α is learned jointly with the whole model using classical gradient-based methods with backpropagation without weight decay to avoid pushing α to zero during the training. From a computational point of view, the additional parameter appears negligible if compared to the total number of network parameters when an α-sharing policy is used. Empirical experiments show that the magnitude of α rarely is larger than 1, although no constraints on its range are applied. However, the resulting function remains basically a modified version of the ReLU function that can change its shape just in the negative part.

Parametric ELU. Trottier, Gigu, Chaib-draa, et al. (2017) try to eliminate the need to manually set the α parameter of ELU unit by proposing an alternative version based on two trainable parameters, i.e.

$$\text{PELU}(a) = \begin{cases} \frac{\beta}{\gamma} a & \text{if } a \geq 0 \\ \beta \cdot (\exp(\frac{a}{\gamma}) - 1) & \text{otherwise.} \end{cases}$$

where $\beta, \gamma > 0$ control the function shape and are learned together with the other network parameters using some optimization gradient-based method.

Flexible ReLU. Authors in Qiu, Xu, and Cai (2018) propose the following function:

$$\text{frelu}(a) = ReLU(a+\alpha) + \beta$$

with α, β learned by data. This is done to capture the negative information which is lost with the classic ReLU function and to provide the zero-like property (Clevert et al., 2016). Considering that the value of a is a weighted sum of inputs and bias, the α parameter can be viewed as part of the function input, so the authors reduced the function to

$$\text{frelu}(a) = ReLU(a) + \beta.$$

Swish. In Ramachandran et al. (2018) the authors propose a search technique for activation functions. In a nutshell, a set of candidate activation functions is built by combining functions belonging to a predefined set of basis activation functions. For each candidate activation function, a network which uses generated function is trained on some task to evaluate the performance. Between all the tested functions, the best one results to be:

$$\text{Swish}(a) = a \cdot \text{sig}(\alpha \cdot a)$$

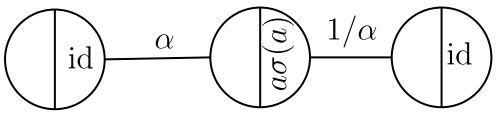

Fig. 6. A possible representation of the Swish activation function using just feed forward neural network layers. Connection with same labels is intended to be shared, while connection with numeric labels is intended to be fixed during the training. See text for further details.

where α is a trainable parameter. it is worth pointing out that for $\alpha \to +\infty$, Swish behaves as ReLU, while, if $\alpha = 1$, it becomes equal to SiLU (Elfwing et al., 2018).

Discussion

The activation functions proposed in this section share the peculiarity to be based on standard fixed activation functions whose shape is tuned using one or more trainable parameters. However, the trained activation function shape turns out to be very similar to its corresponding non-trainable version, with the result of a poor increase of its expressiveness. For example, see Fig. 5, AGSig (AGTanh) function results to be the sigmoid (Tanh) function with smoothness and amplitude tuned by the α and β parameters. Similarly, Swish can be viewed as a parameterized SiLU/ReLU variation whose final shape is learned to have a good trade-off between these two functions. In the end, the general function shape remains substantially bounded to assume the basic function(s) shape on which it has been built.

However, being the most significant part of these functions just a weighted output of the respective weighted input of a fixed activation function, one can notice that it is possible to model each of them by a simple shallow neural subnetwork composed of few neurons. As an illustrative example, consider a neuron n with the Swish activation function and weighted sum of inputs a, n's behavior is equivalent to a feed forward subnetwork composed of three neurons n_1, n_2, n_3 with respective weighted sums of inputs a_1, a_2, a_3 (see Fig. 6): n_1 corresponds to the neuron n equipped with the identity function instead of Swish, n_2 is a neuron equipped with fixed activation function $f_2(a_2) = a_2\sigma(a_2)$ without bias value, and the last neuron n_3 has, again, the identity as activation function and no bias associated. With the connection going from n_1 to n_2 a weight with value α is associated, learned by data together with the other network parameters. The weight α assumes the role of the parameter α of the Swish function, while the connection between n_2 and n_3 has to be constrained to assume the value $\frac{1}{\alpha}$. In fact, the weighted sum fo the inputs of n_2 is $a_2 = \alpha a_1$ and the n_3 final output results to be $\frac{1}{\alpha}a_2 \cdot \sigma(a_2) = \frac{1}{\alpha}\alpha \cdot a_1 \cdot \sigma(\alpha \cdot a_1) = a_1 \cdot \sigma(\alpha a_1)$. Since $a_1 = a$, n_3 output will be equal to $Swish(a)$.

Similar consideration can be made for other trainable activation functions, some of which we report in Figs. 7 and 8.

Another aspect that is worth to stress is that for all the above mentioned trainable activation functions a gradient descent method based on backpropagation was used to train all the parameters of the networks; however, to train the activation function parameters the standard backpropagation formulas have to be suitable and specifically adapted to each trainable activation function.

4.2. Functions based on ensemble methods

With the expression "ensemble methods" we refer to any technique merging together different functions. Basically, each of these techniques uses:

- a *set of basis functions*, which can contain fixed-shape functions or trainable functions or both;
- a *combination model*, which defines how the basis functions are combined together.

As a significant example of this type of approach, in Ramachandran et al. (2018) a method to investigate activation functions built as compositions of several unary and binary functions is proposed, together with a search technique which works in these spaces. In this framework, a set of basis functions is provided (e.g., $\{\sin(a), \max(a, 0), \ldots\}$) together with the number of inputs (unary, binary, ...) of each function. In this case, the combination model is the structure that the final activation function must follow in terms of function compositions (for example, the final output is given by the output of a binary function which has as input two unary functions and so on). The authors state that several models (that they called *search spaces*) have been tried using different combinations of unary and binary functions. Among all the models investigated, we mention *Swish* function, which has been discussed in 4.1. A similar methodology was made in Basirat, Jammer, and Roth (2019), but using a genetic algorithm to learn the best activation function.

However, currently, a significant subset of works based on ensemble methods privileges the use of unary functions combined linearly. In other terms, it is possible to identify a relevant subfamily of works sharing the common feature that the activation functions proposed can be expressed in terms of a linear combination of one-to-one functions, that is constraining the basis functions set to be composed of only functions of the type $g \in \mathbb{R}$, and the combination model to be a linear model. In the remainder of this subsection we focus on this particular subclass of ensemble functions, highlighting how these can be easily expressed as sub-networks, and easily integrated into the model without constructing ad-hoc neurons, and without the need of modifying the learning algorithm, but only using classical feed-forward neural networks layers.

Linear combination of one-to-one functions

In this section we will group all the works which present trainable activation functions which can be expressed as a combination of several one-to-one functions. In a nutshell, the resulting trainable activation function can be reduced to $f(a) = \sum_{i=1}^{h} \alpha_i \cdot g_i(a)$ where $g_1, g_2, \ldots, g_h$ are one-to-one mapping functions, that is $g_i : \mathbb{R} \to \mathbb{R}$, $\forall 1 \leq i \leq h$ and α_i are weights associated with the functions, usually learned by data. Thus, in this type of approach the combination model is the linear combination and the basis functions are one-to-one mapping functions.

All the works reviewed in this section can be reduced to this general form. However, we distinguish at least two different ways in which they are presented originally: the former is based on the fact that the linear combination of functions can be expressed as a neural network; in the latter, the trainable activation functions are expressed using an analytical form. However, we will see that also for several of these functions a simple equivalent formulation in terms of subnetworks exists. In the next part of this section, we will review the main works, dividing them according to the way they are originally presented.

Adaptive activation functions. In Qian et al. (2018) the authors present, in a probabilistic and hierarchical context, different mixtures of the ELU and ReLU functions able to obtain a final activation function learned from data. The authors propose the following activation functions:

- Mixed activation: $\Phi_M(a) = p \cdot \text{LReLU}(a) + (1 - p) \cdot \text{ELU}(a)$ with $p \in [0, 1]$ learned from data;

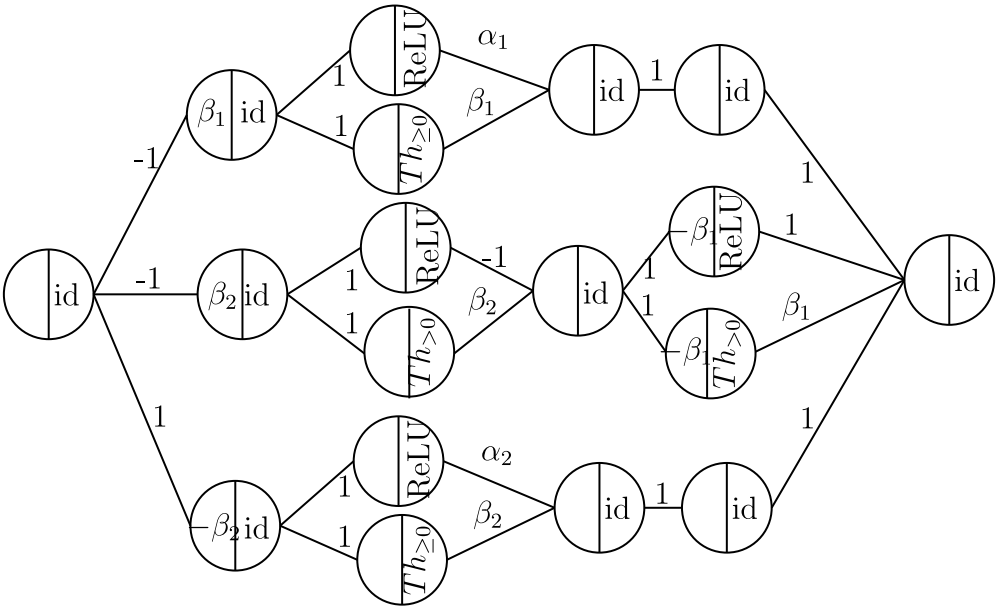

Fig. 7. A possible representation of the S-Shaped ReLU activation function using just feed forward neural network layers. Connection with same labels is intended to be shared, while connection with numeric labels is intended to be fixed during the training. See text for further details.

- Gated activation: $\Phi_G(a) = \text{sig}(\beta a) \cdot \text{LReLU}(a) + (1 - \text{sig}(\beta a)) \cdot \text{ELU}(a)$ with $\text{sig}(\cdot)$ the sigmoid function and β learned from data;
- Hierarchical activation: this function is composed of a three-level subnetwork, where each input unit u is connected to n units, and every pair of nodes is combined similarly as in gated activation (with the substitution of the ELU functions with PELU and ReLU with PReLU). At the same time, the last layer takes the maximum of the middle-level unit. So, the hierarchical organization can be formalized as follows:

$$\begin{cases} \phi_{l,i}^{(1)}(a) = \text{PReLU}(a) \text{ and } \phi_{r,i}^{(1)}(a) = \text{PELU}(a) \\ \phi_i^{(2)}(a) = \text{sig}(\beta_i a) \cdot \phi_{l,i}^{(1)}(a) + (1 - \text{sig}(\beta_i a)) \cdot \phi_{r,i}^{(1)}(a) \\ \phi^{(3)}(a) = \max_i \phi_i^{(2)}(a) \end{cases}$$

  where $\phi_{l,i}^{(1)}(a) =$ is for the units of the first level, with $i = 1, 2, \cdot, m$, $\phi_i^{(2)}(a)$ is used for the ith unit of the second level, and $\phi^{(3)}(a)$ is for the third level; the final activation function is $\Phi_H(a) = \phi_l^{(3)}(a)$.

Variable activation function. In Apicella et al. (2019) trainable activation functions are expressed in terms of sub-networks with only one hidden layer, relying on the consideration that a one-hidden layer neural network can approximate arbitrarily well any continuous functional mapping from one finite-dimensional space to another, enabling the resulting function to assume "any" shape. In a nutshell, the proposed activation is modeled as a non-linear activation function f with a neuron with an Identity activation function which sends its output to a one-hidden-layer sub-network with just one output neuron having, in turn, an Identity as an output function.

The proposed model can be formalized as

$$VAF(a) = \sum_{j=1}^{k} \beta_j g(\alpha_j a + \alpha_{0j}) + \beta_0 \tag{1}$$

where g is a fixed-shape activation function, k a hyper-parameter that determines the number of hidden nodes of the subnetwork and α_j, α_{0j}, β_j, β_0 are parameters learned from the data during the training process.

Kernel-based activation function. The activation function proposed in Scardapane et al. (2019) is modeled in terms of a kernel expansion:

$$KAF(a) = \sum_{i=1}^{D} \alpha_i k(a, d_i) \tag{2}$$

where $\alpha_1, \alpha_2, \ldots, \alpha_D$ are the trainable parameters, k a kernel function $k : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ and $d_1, d_2, \ldots, d_D$ are the dictionary elements, sampled from the real line for simpleness. The choice of the kernel function is widely discussed in Scardapane et al. (2019). Furthermore, in Scardapane, Van Vaerenbergh, Hussain, and Uncini (2018), the authors show two different methods to make KAF adapt to be used also with Complex-valued Neural Networks (Nitta, 2013).

Adaptive blending unit. The work proposed in Sütfeld, Brieger, Finger, Füllhase, and Pipa (2018) combines together a set of different functions in a linear way, that is:

$$\text{ABU}(a) = \sum_{i=1}^{k} \alpha_i \cdot f_i(a)$$

with $(\alpha_0, \alpha_1, \alpha_2, \ldots, \alpha_k)$ parameters to learn and $(f_1(\cdot), f_2(\cdot), \ldots, f_k(\cdot))$ a set of activation functions that is, in the original study, composed of tanh, ELU, ReLU, id, and Swish. The α parameters are all initialized with $\frac{1}{k}$ and are constrained using four different normalization schemes.

Adaptive piecewise linear units. In Agostinelli, Hoffman, Sadowski, and Baldi (2015) the activation functions are modeled as a sum of hinge-shaped functions that results in a different piece-wise linear activation function for every neuron:

$$\text{APL}(a) = \max(0, a) + \sum_{i=1}^{k} w_k \max(0, -a + b_k)$$

where k is a hyper-parameter and w_k, b_k are parameters learned during the network training. The authors show that APL can learn convex and non-convex functions. However, the total overhead in terms of number of parameters to learn compared with a classic NN with n units is $2 \cdot k \cdot n$, so the number of parameters

increases with the number of hidden units and, for a large input, the learned function tends to behave as a ReLU function, reducing the expressiveness of the learned activation functions. In the experiments reported in Agostinelli et al. (2015) the value of k was determined using a validation process, while the w and b parameters were regularized with an L_2 penalty, so that the optimizer can avoid numerical instability leaving out too large values for the parameters.

Harmon & Klabjan activation ensembles. Some studies try to define activation functions using different available activation functions rather than creating an entirely new function. For example, in Klabjan and Harmon (2019) the authors allow the network to choose the best activation function from a predefined set $F = \{f^{(1)}, f^{(2)}, \ldots, f^{(k)}\}$, or some combination of those. Differently from Maxout (see Section 5), the activation functions are combined together instead of simply taking the function with the maximum value.

The activation function proposed by Klabjan and Harmon (2019) works on single mini-batch, i.e. its input is tuple $\mathbf{a}^{(u)} = (a_1^{(u)}, a_2^{(u)}, \ldots, a_B^{(u)})$ where every $a_i^{(u)}$, $1 \leq i \leq B$ refers to the unit u on the ith element of the mini-batch. The proposed activation function is based on a sum of normalized functions weighted by a set of learned weights; the resulting activation function $\Phi(a)$ of $\mathbf{a}_b^{(u)}$ has the form:

$$\Phi^{(u)}(a_b^{(u)}) = \sum_{j=1}^{k} \alpha_u^j \left(\eta^{(j)} g^j(a_b^{(u)} + \delta^{(j)})\right)$$

where α_u^j is a weight value for the uth unit and the jth function, $\eta^{(j)}$ and $\delta^{(j)}$ are inserted to allow the network choosing to leave the activation in its original state if the performance is particularly good and

$$g^j(a_b^{(u)}) = \frac{f^j(a_b^{(u)}) - \min_i f^j(a_i^{(u)})}{\max_i f^j(a_i^{(u)}) - \min_i f^j(a_i^{(u)}) + \epsilon}$$

where ϵ is a small number. The authors emphasize that, during the experiments, many neurons favored the ReLU function since the respective $a_i^{(u)}$ had greater magnitude compared with the others. The learning of the α values was done in terms of an optimization problem with the additional constraint that α values must be non-zero and sum to one to limit the magnitude. So, the approach proposed by Klabjan and Harmon (2019) seems to require additional computational costs due to the resolution of a new optimization problem together with the standard network learning procedure.

S-shaped ReLU. Taking inspiration from the Weber–Fechner law (Fechner, 1966) and Stevens law (Stevens, 1957), the authors of Jin et al. (2016) designed an activation S-shape function determined by three linear functions:

$$\text{SReLU}(a) = \begin{cases} \beta_1 + \alpha_1(a - \beta_1) & \text{if } a \leq \beta_1 \\ a & \text{if } \beta_1 < a < \beta_2 \\ \beta_2 + \alpha_2(a - \beta_2) & \text{if } a \geq \beta_2 \end{cases}$$

where b_1, w_1, b_2, w_2 are parameters that can be learned together with the other network parameters. Also, in this case, the weight decay cannot be used during the learning because it tends to pull the parameters to zero. SReLU can learn both convex and non-convex functions, differently from other trainable approaches like Maxout unit (Goodfellow, Warde-Farley, Mirza, Courville, & Bengio, 2013) that can learn just convex function. Furthermore, this function can approximate also ReLU when $b_2 \geq 0, w_2 = 1, b_1 = 0, w_1 = 0$ or LReLU/PReLU when $b_2 \geq 0, w_2 = 1, b_1 = 0, w_1 > 0$.

Discussion

As stated above, a linear combination of one-to-one functions is always expressible as a linear combination of one-to-one mappings. Due to this common property, several of them can be expressed in terms of feed-forward neural networks. Some of these functions, as VAF and Adaptive Activation Functions, are already modeled as sub-networks that can be integrated into the main architecture without changing the learning algorithm in the respective presentation works. The sub-network structure allows the function to be trained in a "transparent" way for the rest of the network and the chosen training algorithm. On the other hand, several of the remaining functions are analytically presented by the authors instead of using the neural network paradigm. However, for several of these functions we show that a natural equivalent formulation in terms of subnetworks exists, making these architectures not only easy to integrate into the main models, but also easier to study using the general rules of feed-forward neural networks. We report some of the equivalent models in Figs. 6–8. For instance, the SReLU can be expressed as

$$\begin{aligned} SReLU(a) = & -\alpha_1 ReLU(\beta_1 - a) + \beta_1 Th_{\geq 0}(\beta_1 - a) \\ & + \beta_1 Th_{>0}\big(\beta_2 Th_{>0}(\beta_2 - a) - ReLU(\beta_2 - a) - \beta_1\big) \\ & + ReLU\big(\beta_2 Th_{>0}(\beta_2 - a) - ReLU(\beta_2 - a) - \beta_1\big) \\ & + \alpha_2 ReLU(\beta_2 - a) + \beta_2 Th_{\geq 0}(\beta_2 - a) \end{aligned}$$

where $Th_{>0}(x) = 1 - Th_{\geq 0}(-x)$ and $\beta_1 \geq \beta_2$. So the SReLU functions (and all the others 1-to-1 functions) can be expressed as FFNN with constraints on the parameters and the inner activation functions.

4.3. Outliers

In Piazza et al. (1992) an attempt to model activation functions using a polynomial activation function with adaptable coefficients was proposed. For a given degree $k \in \mathbb{N}$, the relative polynomial function is

$$\text{AP}(a) = \sum_{i=0}^{k} \alpha_i \cdot a^i$$

with $(\alpha_0, \alpha_1, \alpha_2, \ldots, \alpha_k)$ parameters to learn. These function parameters can be learned together with the network parameters using gradient descent with back-propagation. However, it must be taken into account that networks with polynomial activation functions are not universal approximator, as shown in Stinchcombe and White (1990).

In the context of fuzzy activation functions (Jou, 1991; Karakose & Akin, 2004; Nguyen & Korikov, 2017), in Beke and Kumbasar (2019a, 2019b) a neural unit based on Type-2 fuzzy logic is proposed, which is capable of representing simple or sophisticated activation functions through three hyperparameters defining the slopes in the negative and positive quadrants. These hyperparameters can be set as adjustable parameters which can be learned together with the other network ones.

Eisenach, Wang, and Liu (2016) try to approximate an activation function using a Fourier expansion. This study uses a 2-stage Stochastic Gradient Descent (SGD) algorithm to learn the parameters of the activation functions and of the network. Urban, Basalla, and van der Smagt (2017) try to learn the activation function using Gaussian processes while Goh and Mandic (2003) learn the amplitude of the activation functions in Recurrent Neural Networks (Cardot & Romuald, 2011). Ertuğrul (2018) proposes two trainable activation functions using linear regression, but using network architectures different from the classic Feed-Forward Neural Networks (see for example Chen, Song, Li, Yang, & Wu, 2018; Huang, 2015).

Other studies propose activation functions whose shape is computed using interpolation techniques. These techniques may need some additional input, depending on the technique used (for example a set of sampled points from a start function).

In Guarnieri (1995) the authors introduced the use of spline based activation functions whose shape can be learned by data using a set of Q representative points. This method has been improved by Scardapane, Scarpiniti, Comminiello, and Uncini (2017) and Vecci, Piazza, and Uncini (1998). More in detail, this technique tries to find a cubic spline to model the activation function sampling the control points from a sigmoid (as in Guarnieri, 1995) or from another function (e.g. hyperbolic tangent, as in Scardapane et al., 2017) assuring universal approximation capability. The resulting function can be expressed as follows:

$$\mathrm{SAF}(a) = \mathbf{u}^T B \mathbf{q}_{i:i+P}$$

where:

- i is the index of the closest knot;
- $\mathbf{q}$ is the knots vector, with $\mathbf{q}_{i:i+P} := (q_i, q_{i+1}, \ldots, q_{i+P})^T$, so the output is computed by spline interpolation over the closest knot and its P right-most neighbors. Supposing that the knots are uniformly spaced, i.e. $\mathbf{q}_{i+1} = \mathbf{q}_i + \Delta t$, for a fixed $\Delta t \in \mathbb{R}$, the normalized abscissa value can be computed as $u = \frac{a}{\Delta t} - \lfloor \frac{a}{\Delta t} \rfloor$;
- $\mathbf{u}^T = (u^P, u^{P-1}, \ldots, u, 1) \in \mathbb{R}^{P+1}$ is the reference vector;
- $B \in \mathbb{R}^{(P+1)\times(P+1)}$ is the basis spline matrix. Different bases make different interpolation schemes; in Vecci et al. (1998) the authors used the Catmull-Rom matrix (Smith, 1983).

The $\mathbf{q}$ values are then adapted during the learning procedure, adding a regularization term on $\mathbf{q}$ to prevent the over-fitting. The regularization term results to be a very critical issue: while the authors of Vecci et al. (1998) act on the Δx value, in Scardapane et al. (2017) the authors proposed to penalize changes in $\mathbf{q}$ compared with a "good" set of values, as for example the initial control points values, sampled from a standard NN activation function.

Based on a similar principle, in Wang, Liu, and Foroosh (2018) the authors introduce Look-up Table Unit based on spline; in this work, the activation function is controlled by a look-up table containing a set of anchor-points that control the function shape. The look-up table idea is not new in trainable activation function field; a first example in this direction is found in Piazza et al. (1993), where a generic adaptive look-up table was addressed by the neuron linear combination and learned by data. The main difference with Wang et al. (2018) is in the structure of the look-up table. It now returns the result of a spline interpolation instead of the raw number in the table. More in detail, defining the set of anchor point as $A = \{(q_1, u_1), (q_2, u_2), \ldots, (q_m, q_m)\}$ with $q_i = q_1 + i \cdot \Delta t$ and u_i the trainable parameters, Wang et al. (2018) propose two different methods to generate the activation function; the first one by interpolation, which results in the function:

$$\mathrm{LuTU}(a) = \frac{1}{t} u_i (q_{i+1} - a) + u_{i+1}(a - q_i), \text{ if } q_i \le a \le q_{i+1}$$

and the second one using cosine smoothing:

$$\mathrm{LuTU}(a) = \sum_i^m u_i \cdot r(a - q_i, \alpha t)$$

where $\alpha \in \mathbb{N}$ and

$$r(w, \tau) = \begin{cases} \frac{1}{2\tau}(1 + \cos(\frac{\pi}{\tau} w)) & \text{if } -\tau \le w \le \tau \\ 0 & \text{otherwise.} \end{cases}$$

The method based on cosine smoothing was proposed to address the gradient instability suffered by the interpolation method. This kind of approaches requires to set additional hyper-parameters like the function input domain, the number of anchor points and the space between them, the α value in Wang et al. (2018) second approach or the spline type for (Scardapane et al., 2017; Vecci et al., 1998) approaches. Beyond a robust mathematical formulation, these methods seem to require the tuning of several hyper-parameters.

5. Trainable non-standard neuron definitions

So far, we took care of activation functions in the classic meaning given by literature, i.e., a function $o(a)$ that builds the output of the neuron using as input the value returned by the internal transformation $\mathbf{wx}+b$ made by the classic computational neuron model, as described in Section 3. In the remainder of this section, we will review works which change the standard definition of neuron computation but are considered as neural network models with trainable activation functions in the literature. In other terms, these functions can be considered as a different type of computational neuron unit compared with the original computational neuron model.

Maxout unit. Goodfellow et al. (2013) was one of the first modern studies that proposed a new kind of neural unit with a different output computation. The name *Maxout* was given by the fact that the unit output is the max of a set of linear functions. Maxout units should not be considered simply activation function, since they use multiple weighted sums for every neuron instead of a single weighted sum $a = \mathbf{wx} + b$ used with classical artificial neurons. More precisely, a Maxout unit makes a vector $\mathbf{a} = (a_1, a_2, \ldots, a_k)$ with $\forall i \in \{1, k\}$, $a_i = \mathbf{w}^{(i)T}\mathbf{x} + b^{(i)}$ with $\mathbf{x} \in \mathbb{R}^d$ output given by the previous layer, $\{\mathbf{w}^{(1)} \in \mathbb{R}^d, \mathbf{w}^{(2)} \in \mathbb{R}^d, \ldots, \mathbf{w}^{(k)} \in \mathbb{R}^d\}$, $\{b^{(1)} \in \mathbb{R}^m, b^{(2)} \in \mathbb{R}^m, \ldots, b^{(k)} \in \mathbb{R}^m\}$. w_i and b_i are the parameters to be learned. In the end it returns

$$\mathrm{Maxout}(\mathbf{a}) = \max_{1 \le j \le k} \{a_i\}.$$

In other words, Maxout units take the maximum value over a subspace of k trainable linear functions of the same input $\mathbf{x}$, obtaining a piece-wise linear approximator capable of approximating any convex function. The same model can be defined arranging all the $\mathbf{w}^{(i)}$ vectors as column of a single matrix $W \in \mathbb{R}^{d\times k}$ and all the $b^{(i)}$ scalars in a single vector $\mathbf{b} \in \mathbb{R}^k$, obtaining $\mathbf{a} = W^T\mathbf{x} + \mathbf{b}$.

To notice that, setting $k = 2$ and $\mathbf{w}^{(1)} = 0, b^{(1)} = 0$, Maxout becomes

$$\begin{aligned} \mathrm{Maxout}(\mathbf{a}) &= \max(\mathbf{w}^{(1)T}\mathbf{x} + b^{(1)}, \mathbf{w}^{(2)T}\mathbf{x} + b^{(2)}) = \\ &\max(0, \mathbf{w}^{(2)T}\mathbf{x} + b^{(2)}) = \mathrm{ReLU}(w^{(2)T}\mathbf{x} + b^{(2)}), \end{aligned}$$

In a similar way, Maxout unit can be made equivalent to Leaky ReLU, so Maxout can be viewed as a generalization of classic rectifier-based units. Being Maxout constituted by a set of feed-forward sub-networks, its parameters can be learned together with the whole network using classical gradient descent approaches. By running a cross-validation experiment, in Goodfellow et al. (2013) the authors found that Maxout offers a clear improvement over ReLU units in terms of classification errors. Despite the performance, this approach requires many new weights compared with a classic network based on ReLU and classical neural units, namely k times the number of parameters for every single neuron, significantly increasing the cost of the learning process.

Multi-layer maxout. The Maxout-based network is generalized in Sun, Su, and Wang (2018). The authors adopt a function composition approach that they call Multi-layer Maxout Network (MMN) further increasing the number of parameters. To limit the computational costs introduced, the authors proposed to replace just a portion of activation functions in traditional DNN with MMNs as a trade-off scheme between the accuracy and computing resources.

Probabilistic maxout. In Springenberg and Riedmiller (2014) the authors describe Probout, a stochastic generalization of the Maxout unit trying to improve its invariance replacing the maximum operation in Maxout with a probabilistic sampling procedure, i.e.

$$\text{Probout}(\mathbf{a}) = a_i, \text{ with } i \sim \text{Multinomial}(p_1, p_2, \ldots, p_k)$$

where $p_i = \frac{\exp(\lambda a_i)}{\sum_{j=1}^{k} \exp(\lambda a_j)}$, and λ is a hyperparameter. The maximum operation substitution in Maxout arises from the observation that to use other operations could be useful to improve the performances. To notice that the Probout function reduces to Maxout for $\lambda \to +\infty$.

NIN & CIC. The authors of Lin, Chen, and Yan (2013) proposed a trainable activation function designed for Convolutional Neural Networks. In this work, the activation functions of a convolution layer are replaced with a full-connected multilayer perceptron. In the following of this paragraph, we indicate with:

- $X \in \mathbb{R}^{h \times w \times c}$, the input of a CNN;
- $X_{ij} \in \mathbb{R}^{t \times t \times c}$ an input *patch* of size t centered in the position ij, that is the submatrix composed of all the elements of X belonging to the rows $i-t/2, i-t/2+1, \ldots, i+t/2$ and the columns $j-t/2, j-t/2+1, \ldots, j+t/2$ and each channel $1, 2, \ldots, c$.

The Network in Network (NIN) architecture proposed by Lin et al. (2013) is based on MLP sub-nets with l layers nested into a CNN and applied to every patch X_{ij}. This operation results in a set of functions $f^{(i)}$ that can be expressed as:

$$f^{(1)}_{i,j,c_1} = \text{ReLU}(W^{(1)}_{c_1} X_{ij} + \mathbf{b}_{c_1}),$$

$$\vdots$$

$$f^{(l)}_{i,j,c_l} = \text{ReLU}(W^{(l)}_{c_l} f^{(l-1)}_{i,j,*} + \mathbf{b}_{c_l}).$$

where l is the number of layers of the MLP and c_i is the number of the input channels for $i = 1$ or the number of the level filters for every $i > 1$. We indicate with $f^{l-1}_{i,j,*}$ the vector composed of the functions output in the point (x, j) for each channel, i.e. $f^{(l)}_{i,j,*} = (f^{(l)}_{i,j,1}, f^{(l)}_{i,j,2}, \ldots f^{(l)}_{i,j,c_l})$.

So, this subnet can be viewed as an MLP with ReLU units and the output results to be a map constituted by the output of the last layer functions $f^{(l)}$. Despite the good performances obtained, these methods require to learn a lot of extra parameters, especially when l is large. NIN seems to move away from the concept of variable activation function of a single neuron because the MLP could have common connections with other nodes of the previous layers, in other words there are no constraints on the number of the output that the final layer of the MLP can have more than one output.

Based on NIN, authors of Pang, Sun, Jiang, and Li (2017) proposed Convolution in Convolution (CIC) which uses a sparse MLP instead of the classic full-connected MLP as activation function.

Batch-normalized maxout NIN. In Chang and Chen (2015) Maxout and NIN are combined together with Batch Normalization (Ioffe & Szegedy, 2015). The proposed method replaces the ReLU functions present in NIN with Maxout to avoid the zero saturation and adds Batch Normalization to avoid the problems connected with changes in data distribution (Ioffe & Szegedy, 2015).

LWTA. In Srivastava, Masci, Kazerounian, Gomez, and Schmidhuber (2013) an interesting neural subnetwork based on the local competition between several neurons is presented. Relying on biological models of brain processes, a Local Winner-Take-All subnetwork (LWTA) consists in blocks of neurons whose outputs are fed in a *competition/interaction* function that returns the final output of the block neurons. Similarly to Maxout, given a set (block) of k neurons, with activation given by $a_i = \mathbf{w}^{(i)T}\mathbf{x}, \forall i \in \{1, \ldots, k\}$ with $\mathbf{x} \in \mathbb{R}^d$ output given by the previous layer and $\{\mathbf{w}^{(1)} \in \mathbb{R}^d, \mathbf{w}^{(2)} \in \mathbb{R}^d, \ldots, \mathbf{w}^{(k)} \in \mathbb{R}^d\}$ parameters to be learned, the final output of a neuron i is given by a function of all the neurons belonging to the same block of i, that is:

$$LWTA_i(\mathbf{a}) = \begin{cases} a_i & \text{if } a_i = \max(\mathbf{a}) \\ 0 & \text{otherwise,} \end{cases} \quad \forall i \in \{1, \ldots, k\}$$

that is, only the "winner" neuron will propagate its output to the next layer of the network, turning off the activations of the remaining neurons in the same block. In this way, for a given input, only a subset of the network parameters is used. The authors hypothesize that using different subsets of parameters for different inputs allows the architecture to learn more accurately.

Some considerations. We highlight that in some cases, the basic building blocks of a neural network are not the neurons, but groups of neurons, for example, the layers of a multi-layer feed-forward neural network (see, Eckle & Schmidt-Hieber, 2019), and the activation function is expressed as a mapping $f : \mathbb{R}^k \longrightarrow \mathbb{R}^k$, where k is the number neurons in the layer. However, this is a notational or computational simplification, and one can again consider the neuron as the primary building block and, consequently, the activation function as a function $f : \mathbb{R} \longrightarrow \mathbb{R}$. Nevertheless, sometimes the input and activation functions of the basic building blocks differ substantially from the classical ones. Maxout networks (Goodfellow et al., 2013) are a significant example. In the original paper, as above described, the authors propose a Maxout unit which, given an input $\mathbf{x} \in \mathbb{R}^d$, computes its output as $h(\mathbf{x}) = \max_{i=1,\ldots,k}\{a_i\}$, where $a_i = \mathbf{w}^{(i)T}\mathbf{x} + b^{(i)}$. The input function of the maxout unit is a functional mapping $I : \mathbb{R}^d \to \mathbb{R}^k$, and the activation function corresponds to the maximum operator. The authors themselves state that the maxout unit is equivalent to a standard one-hidden-layer feed-forward network (shallow network) with k hidden neurons and one output neuron. The k hidden neurons have input and activation functions corresponding to the standard weighted linear sum of the incoming input values and the identity function, respectively. The output neuron has the maximum as input function, and the identity as activation function. In the literature a Maxout network is often considered a network with trainable activation functions (see, for example, Klabjan & Harmon, 2019; Sun et al., 2018), however its activation function definition does not fit with the one we adopted in this paper. For this reason, in our taxonomy, we excluded the Maxout networks from the class of "trainable activation function", and we include this model into a specific "trainable" subclass of the "Non-standard neuron definitions" class.

Another significant example is Network in Network (NiN) (Lin et al., 2013). This model is introduced in the context of Convolutional Neural Networks (CNN). The convolution filter in a CNN is a Generalized Linear Model (GLM), while in NIN the GLM is replaced by a more potent nonlinear function approximator (Lin et al., 2013). In particular, the authors use as general nonlinear function a multilayer perceptron called "mlpconv". In the literature, this nonlinear filter is often interpreted as a single unit with trainable activation function (e.g., Eisenach et al., 2016; Klabjan & Harmon, 2019). Also, in this case, this interpretation does not fit with the standard activation function definition which we adopted in this survey paper, thus in our taxonomy, we included this model in the same subclass of Maxout model.

6. Performance and experimental architecture comparison

All the reviewed studies highlight the improvements in terms of accuracy compared with using non-trainable activation functions in their model. However, making an exhaustive comparison between all the proposed approaches could be not very significant due to the differences in the experimental setup. Apart from works which make an explicit comparison with other trainable (and not) activation functions (DasGupta & Schnitger, 1993; Karlik & Olgac, 2011; Nwankpa et al., 2018), a complete comparison among all the existing architectures is not straightforward, even if the experiments are usually performed on standard and shared datasets, as SVHN (Netzer et al., 2011), MNIST (LeCun, Bottou, Bengio, & Haffner, 1998), CIFAR10, CIFAR100 (Krizhevsky & Hinton, 2009), ImageNet (Russakovsky et al., 2015). This is due to different choices of neural networks architectures, learning algorithms and hyper-parameters in experimental setups. With this in mind, in order to sketch an overview on this comparison, in Table 2 we show the different architectures and datasets used in the main studies. However, also for studies that use the same architectures, small changes to integrate the proposed model into the setup architectures may have been necessary.

Nevertheless, several works on trainable activation functions proposed in literature report improvements in comparison with equivalent architectures equipped with classical fixed-shape activation functions as ReLU or sigmoid or other trainable models. Usually, this is shown through a comparison between identical setup architectures but different activation functions (see Table 3). However, it is not clear if the performance improvements are due to the learning capability or are simply a consequence of the increased complexity of the final setup architecture.

7. Conclusions

In this paper, we have discussed the most common activation functions presented in the literature, focusing on the trainable ones and proposing a possible taxonomy to distinguish them. We have divided the proposed functions into two main categories: fixed shape and trainable shape. In the second class of activation functions two different families of trainable activation functions have been individuated: *parameterized standard functions* and *functions based on ensemble methods*. The latter has been refined further by isolating another activation function family: *linear combination of one-to-one functions*. This taxonomy shows the great variety of activation functions proposed in literature, and several works on trainable activation functions report substantial performance improvements in comparison with equivalent neural network architectures equipped with classical fixed-shape activation functions as ReLU or sigmoid. However, it must be kept in mind that the activation functions are not the only entities that determine the performances of a neural network. Other hyperparameters, such as the number of neurons and the way they are arranged between them or the weights initialization protocol, can be decisive for the performance of the network, together with the hyper-parameter values required by the learning algorithms. Furthermore, even if the experiments are conducted on the same datasets, these may have been preprocessed in different ways (e.g., ZCA (Bell & Sejnowski, 1997) or data-augmentation (Wang & Perez, 2017)) which can in turn condition the results. So, it can be challenging to make a comparison, in terms of network performances, among the activation functions that have been proposed in literature, since the experiments are often conducted using different experimental setups. Table 3 shows the performances in terms of accuracy reported in several works on the most commonly used datasets. As an indication, we report just the best values obtained, without taking care of the different architectures or experimental setup used in the different works.

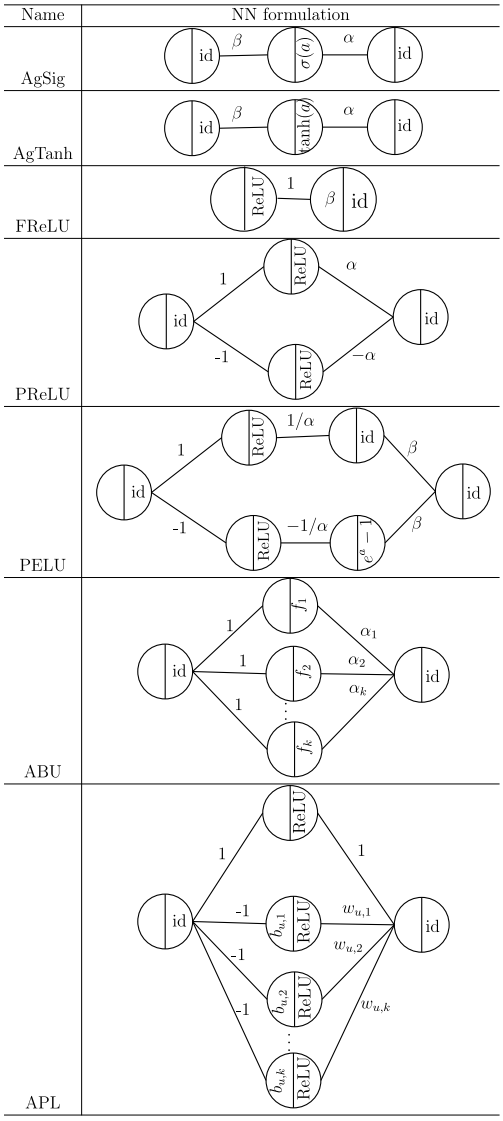

Fig. 8. Several trainable activation functions represented as feed forward layers with constrained weights. The connections having the same labels have shared weights, that is the same value. The connection having numeric values as labels must have the same fixed value during the training.

By a more in-depth analysis, it is important to note that several proposed trainable activation functions can be expressed in terms of linear combination of fixed non-linear functions, so they can be expressed in turn as subnetworks nested in the main architectures, as explicitly reported in the models proposed in Apicella et al. (2019) and Qian et al. (2018). In Figs. 6 7 and 8, we try to draw possible equivalent implementations of some proposed functions in terms of neural (sub)networks architectures. Looking at these functions in this respect, it is easy to see that every neural network architecture equipped with one of these trainable functions can be modeled with an equivalent deeper architecture equipped using just fixed-shape functions. This could lead to the observation that several architectures that use trainable activation functions could reach similar results using deeper architectures possibly imposing some constraints on specific weight layers such as weight sharing without needing trainable activation functions. Thus, neural networks with trainable activation functions can be cheaper in terms of computational complexity

Table 2
The setup architectures used by some works and the dataset used for the experiments made.

Reference paper	Setup architecture(s)	Dataset(s)
Maxout (Goodfellow et al., 2013)	Ownns	MNIST CIFAR 10/100
NIN (Lin et al., 2013)	Ownns	SVHN MNIST CIFAR 10/100
ProbMaxout (Springenberg & Riedmiller, 2014)	Based on Goodfellow et al. (2013)	SVHN CIFAR 10/100
BN-MIN (Chang & Chen, 2015)	Based on Lin et al. (2013)	SVHN MNIST CIFAR 10/100
PRELU (He et al., 2015)	based on VGG-19 (Simonyan & Zisserman, 2015)	ImageNet
APL (Agostinelli et al., 2015)	Based on Srivastava, Hinton, Krizhevsky, Sutskever, and Salakhutdinov (2014)	CIFAR 10/100 Higgs to $\tau^+ - \tau^-$ (Baldi, Sadowski, & Whiteson, 2015)
S-Shaped (Jin et al., 2016)	Based on Lin et al. (2013) and GoogleNet (Szegedy et al., 2015)	MNIST CIFAR 10/100 ImageNet
H.& K. (Klabjan & Harmon, 2019)	Ownns and Lasagne ResNet (http://github.com/Lasagne)	MNIST ISOLET CIFAR 10/100
Eisenach et al. (2016)	Based on Srivastava et al. (2014)	MNIST CIFAR10
Swish (Ramachandran et al., 2018)	Based on ResNets (He, Zhang, Ren, & Sun, 2016; Zagoruyko & Komodakis, 2016), DenseNet (Huang, Liu, Van Der Maaten, & Weinberger, 2017), Inception (Szegedy, Ioffe, Vanhoucke, & Alemi, 2017), MobileNets (Howard et al., 2017; Zoph, Vasudevan, Shlens, & Le, 2018) and Transformer (Vaswani et al., 2017)	CIFAR 10/100 ImageNet, WMT2014
GA/MA/HA (Qian et al., 2018)	Based on Lin et al. (2013)	MNIST CIFAR 10/100 ImageNet
ABU (Sütfeld et al., 2018)	Owns	CIFAR 10/100
FReLU (Qiu et al., 2018)	Owns and based on ResNets (https://github.com/facebook)	CIFAR 10/100
MMN (Sun et al., 2018)	Ownns and based on Lin et al. (2013)	CIFAR 10/100 ImageNet
LUTU (Wang et al., 2018)	Based on ResNets (https://github.com/facebook)	CIFAR10 ImageNet
PELU (Trottier et al., 2017)	Based on ResNets (Shah, Kadam, Shah, Shinde, & Shingade, 2016), (Lin et al., 2013), AllCNN (Springenberg, Dosovitskiy, Brox, & Riedmiller, 2015), Overfeat (Sermanet et al., 2014)	CIFAR 10/100 ImageNet
KAF (Scardapane et al., 2019)	Owns, based on Baldi, Sadowski, and Whiteson (2014) and VGG (Simonyan & Zisserman, 2015)	Sensorless (Dheeru & Karra Taniskidou, 2017) SUSY (Baldi et al., 2014) MNIST Fashion MNIST CIFAR 10/100
VAF (Apicella et al., 2019)	Owns, based on Lin et al. (2013) and Scardapane et al. (2019)	Sensorless (Dheeru & Karra Taniskidou, 2017), MNIST Fashion MNIST CIFAR10 others from UCI (Dheeru & Karra Taniskidou, 2017)
LWTA (Srivastava et al., 2013)	Owns	MNIST Amazon Sentiment Analysis (Blitzer, Dredze, & Pereira, 2007)

Table 3
Accuracy on some datasets of the most common activation functions in literature; here, we report the best values reported without taking into account the differences between the architectures used.

		Accuracy %			
		SVHN	MNIST	CIFAR10	CIFAR100
Fixed shape	*sigmoid*		97.9 (Pedamonti, 2018);		
classic	*tanh*		98.21 (Eisenach et al., 2016)		
	softplus			94.9 (Ramachandran et al., 2018)	83.7 (Ramachandran et al., 2018)
Rectifier-based	ReLU		98.0 (Pedamonti, 2018); 99.53 (Jin et al., 2016); 99.16 (Eisenach et al., 2016); 99.1 (Apicella et al., 2019) 99.15 (Scardapane et al., 2019)	87.55 (Xu et al., 2015); 92.27 (Jin et al., 2016); 94.59 (Trottier et al., 2017); 91.51 (Qian et al., 2018); 84.8 (Eisenach et al., 2016); 95.3 (Ramachandran et al., 2018); 85.7 (Apicella et al., 2019)	57.1 (Xu et al., 2015); 67.25 (Jin et al., 2016); 75.45 (Trottier et al., 2017); 64.42 (Qian et al., 2018) 83.7 (Ramachandran et al., 2018)
	LReLU		98.2 (Pedamonti, 2018); 99.58 (Jin et al., 2016);	88.8 (Xu et al., 2015); 92.3 (Jin et al., 2016); 92.32 (Qian et al., 2018); 95.6 (Ramachandran et al., 2018)	59.58 (Xu et al., 2015); 67.3 (Jin et al., 2016): 64.72 (Qian et al., 2018); 83.3 (Ramachandran et al., 2018)
	RReLU			88.81 (Xu et al., 2015)	59.75 (Xu et al., 2015)
	ELU		98.3 (Pedamonti, 2018);	93.45 (Clevert et al., 2016); 94.01 (Trottier et al., 2017); 92.16 (Qian et al., 2018); 94.4 (Ramachandran et al., 2018)	75.72 (Clevert et al., 2016); 74.92 (Trottier et al., 2017); 64.06 (Qian et al., 2018); 80.6 (Ramachandran et al., 2018)
	AGSig (Swish-1)			95.5 (Ramachandran et al., 2018)	83.8 (Ramachandran et al., 2018)
Learnable shape					
Quasi fixed	PReLU		99.59 (Jin et al., 2016)	88.21 (Xu et al., 2015); 92.32 (Jin et al., 2016); 94.64 (Trottier et al., 2017); 95.1 (Ramachandran et al., 2018)	58.37 (Xu et al., 2015); 67.33 (Jin et al., 2016); 74.5 (Trottier et al., 2017); 81.5 (Ramachandran et al., 2018)
	PELU			94.64 (Trottier et al., 2017)	75.45 (Trottier et al., 2017)
Interpolated	LuTU			94.22 (Wang et al., 2018)	
Ensembled	Gated Act.			92.65 (Qian et al., 2018)	65.75 (Qian et al., 2018)
	Mixed Act.			92.6 (Qian et al., 2018)	65.44 (Qian et al., 2018)
	hierarc. Act.			92,99 (Qian et al., 2018)	66.23 (Qian et al., 2018)
	Harmon Klabjan		99.40 (Klabjan & Harmon, 2019)		74.20 (Klabjan & Harmon, 2019)
	SReLU		99.65 (Jin et al., 2016)	93.02 (Jin et al., 2016)	70.09 (Jin et al., 2016)
	APL		99.31 (Scardapane et al., 2019)	92.49 (Agostinelli et al., 2015)	69.17 (Agostinelli et al., 2015)
	NPF		99.31 (Eisenach et al., 2016)	86.44 (Eisenach et al., 2016)	
	Swish			95.5 (Ramachandran et al., 2018)	83.9 (Ramachandran et al., 2018)
	VAF		99.5 (Apicella et al., 2019)	81.2 (Apicella et al., 2019)	
	KAF		99.43 (Scardapane et al., 2019) 99.5 (Apicella et al., 2019)	84.0 (Scardapane et al., 2019) 80.2 (Apicella et al., 2019)	52.0 (Scardapane et al., 2019)
Changing T.F.	*Maxout*	97.53 (Goodfellow et al., 2013)	99.55 (Goodfellow et al., 2013)	90.62 (Goodfellow et al., 2013)	61.43 (Goodfellow et al., 2013)
	Probout	97.61 (Springenberg & Riedmiller, 2014)		88.65 (Springenberg & Riedmiller, 2014)	61.86 (Springenberg & Riedmiller, 2014)
	MMN			92,34 (Sun et al., 2018)	66.76 (Sun et al., 2018)
	NIN	97.65 (Lin et al., 2013)	99.53 (Lin et al., 2013) 99.6 (Apicella et al., 2019)	91.19 (Lin et al., 2013) 76.3 (Apicella et al., 2019)	64.32 (Lin et al., 2013)
	CIC			91.54 (Pang et al., 2017)	68.6 (Pang et al., 2017)
	MIN	98.19 (Chang & Chen, 2015)	99.76 (Chang & Chen, 2015)	92.15 (Chang & Chen, 2015)	71.14 (Chang & Chen, 2015)
	LSWA		99.43 (Srivastava et al., 2013)		

since there are fewer parameters to control (as weights with fixed values), on the other hand, the performances in accuracy terms are potentially comparable to deeper architectures without trainable activation functions. It is worth to point out that some authors, see for example Apicella et al. (2019) and Scardapane et al. (2019), suggest defining trainable activation functions able to satisfy several desirable properties which include a high expressive power of the trainable activation functions, no external parameter or learning process in addition to the classical ones for neural networks, and the possibility to use classical regularization methods. These properties can be easily satisfied when trainable activation functions are explicitly expressed in terms of subnetworks. Notice that we are not proposing to substitute, in general, trainable activation functions with their corresponding equivalent representations in terms of sub-networks. On the other hand, we think that these equivalent representations might better emphasize similarities and differences among the various trainable activation functions insofar as they are represented using the same formalism, and all the functional properties remain unchanged during the learning stage as they share both the same parameters and the same input–output functional relationship. It is worth pointing out that, however, the equivalent representations of trainable activation functions as sub-networks could be more expensive in terms of computational costs.

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.

Acknowledgments

This research has received funding from EU H2020 under the Specific Grant Agreements No. 785907 and 945539 (Human Brain Project SGA2 and SGA3), and from EU H2020-EIC-FET-PROACT-2019 Grant Agreement No. 951910, MAIA project.

References

Agostinelli, F., Hoffman, M. D., Sadowski, P. J., & Baldi, P. (2015). Learning activation functions to improve deep neural networks. In *3rd international conference on learning representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, workshop track proceedings*.

Alcaide, E. (2018). E-swish: Adjusting activations to different network depths. arXiv preprint arXiv:1801.07145.

Apicella, A., Isgrò, F., & Prevete, R. (2019). A simple and efficient architecture for trainable activation functions. *Neurocomputing*, *370*, 1–15.

Baldi, P., Sadowski, P., & Whiteson, D. (2014). Searching for exotic particles in high-energy physics with deep learning. *Nature Communications*, *5*, 4308.

Baldi, P., Sadowski, P., & Whiteson, D. (2015). Enhanced higgs boson to $\tau+\tau-$ search with deep learning. *Physical Review Letters*, *114*(11), Article 111801.

Basirat, M., Jammer, A., & Roth, P. M. (2019). The quest for the golden activation function. In *Austrian robotics workshop and OAGM workshop*.

Beke, A., & Kumbasar, T. (2019a). Interval type-2 fuzzy systems as deep neural network activation functions. In *2019 conference of the international fuzzy systems association and the European society for fuzzy logic and technology (EUSFLAT 2019)*. Atlantis Press.

Beke, A., & Kumbasar, T. (2019b). Learning with type-2 fuzzy activation functions to improve the performance of deep neural networks. *Engineering Applications of Artificial Intelligence*, *85*, 372–384.

Bell, A. J., & Sejnowski, T. J. (1997). The "independent components" of natural scenes are edge filters. *Vision Research*, *37*(23), 3327–3338.

Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, *5*(2), 157–166.

Bishop, C. (2006). *Pattern recognition and machine learning*. Springer.

Bishop, C. M., et al. (1995). *Neural networks for pattern recognition*. Oxford university press.

Blitzer, J., Dredze, M., & Pereira, F. (2007). Biographies, bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification. In *Proceedings of the 45th annual meeting of the association of computational linguistics* (pp. 440–447).

Cardot, H., & Romuald, B. (2011). *Recurrent neural networks for temporal data processing*. IntechOpen.

Chandra, P., & Singh, Y. (2004). An activation function adapting training algorithm for sigmoidal feedforward networks. *Neurocomputing*, *61*, 429–437.

Chang, J.-R., & Chen, Y.-S. (2015). Batch-normalized maxout network in network. arXiv preprint arXiv:1511.02583.

Chen, F.-C. (1990). Back-propagation neural networks for nonlinear self-tuning adaptive control. *IEEE Control Systems Magazine*, *10*(3), 44–48.

Chen, C.-T., & Chang, W.-D. (1996). A feedforward neural network with function shape autotuning. *Neural Networks*, *9*(4), 627–641.

Chen, S., Cowan, C. F., & Grant, P. M. (1991). Orthogonal least squares learning algorithm for radial basis function networks. *IEEE Transactions on Neural Networks*, *2*(2), 302–309.

Chen, Y., Song, S., Li, S., Yang, L., & Wu, C. (2018). Domain space transfer extreme learning machine for domain adaptation. *IEEE Transactions on Cybernetics*.

Chieng, H. H., Wahid, N., Pauline, O., & Perla, S. R. K. (2018). Flatten-t swish: a thresholded relu-swish-like activation function for deep learning. *International Journal of Advances in Intelligent Informatics*, *4*(2), 76–86.

Clevert, D., Unterthiner, T., & Hochreiter, S. (2016). Fast and accurate deep network learning by exponential linear units (elus). In *4th international conference on learning representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference track proceedings*.

Cybenko, G. (1988). *Continuous valued neural networks with two hidden layers are sufficient*: *Tech. rep.*, Department of Computer Science, Tufts University.

Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, *2*(4), 303–314.

DasGupta, B., & Schnitger, G. (1993). The power of approximating: a comparison of activation functions. In *Advances in neural information processing systems* (pp. 615–622).

Dheeru, D., & Karra Taniskidou, E. (2017). UCI machine learning repository.

Duch, W., & Jankowski, N. (1999). Survey of neural transfer functions. *Neural Computing Surveys*, *2*, 163–213.

Dugas, C., Bengio, Y., Bélisle, F., Nadeau, C., & Garcia, R. (2000). Incorporating second-order functional knowledge for better option pricing.. In *NIPS* (pp. 472–478).

Eckle, K., & Schmidt-Hieber, J. (2019). A comparison of deep networks with ReLU activation function and linear spline-type methods. *Neural Networks*, *110*, 232–242.

Eisenach, C., Wang, Z., & Liu, H. (2016). Nonparametrically learning activation functions in deep neural nets. Online Available.

Eldan, R., & Shamir, O. (2016). The power of depth for feedforward neural networks. In *Conference on learning theory* (pp. 907–940).

Elfwing, S., Uchibe, E., & Doya, K. (2018). Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*.

Ertuğrul, Ö. F. (2018). A novel type of activation function in artificial neural networks: Trained activation function. *Neural Networks*, *99*, 148–157.

Fechner, G. (1966). *Elements of psychophysics*. New York: Holt, Rinehart and Winston.

Glorot, X., Bordes, A., & Bengio, Y. (2011). Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics* (pp. 315–323).

Goh, S. L., & Mandic, D. P. (2003). Recurrent neural networks with trainable amplitude of activation functions. *Neural Networks*, *16*(8), 1095–1100.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press.

Goodfellow, I., Warde-Farley, D., Mirza, M., Courville, A., & Bengio, Y. (2013). Maxout networks. In *Proceedings of Machine Learning Research: vol. 28, Proceedings of the 30th international conference on machine learning* (pp. 1319–1327).

Guarnieri, S. Multilayer neural networks with adaptive spline-based activation functions. In *Proceedings of word congress on neural networks WCNN'95, Washington, DC, July* (pp. 17–21).

Gulcehre, C., Moczulski, M., Denil, M., & Bengio, Y. (2016). Noisy activation functions. In *International conference on machine learning* (pp. 3059–3068).

Hagan, M. T., Demuth, H. B., & Beale, M. (1996). *Neural network design*. Boston, MA, USA: PWS Publishing Co., ISBN: 0-534-94332-2.

Haykin, S. (1994). *Neural networks: a comprehensive foundation*. Prentice Hall PTR.

He, K., Zhang, X., Ren, S., & Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision* (pp. 1026–1034).

He, K., Zhang, X., Ren, S., & Sun, J. (2016). Identity mappings in deep residual networks. In *European conference on computer vision* (pp. 630–645). Springer.

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

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., et al. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861.

Hu, Z., & Shao, H. (1992). The study of neural network adaptive control systems.. *Control and Decision*, *7*(2), 361–366.

Huang, G.-B. (2015). What are extreme learning machines? Filling the gap between frank rosenblatt's dream and john von Neumann's puzzle. *Cognitive Computation*, *7*(3), 263–278.

Huang, G., Liu, Z., Van Der Maaten, L., & Weinberger, K. Q. (2017). Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 4700–4708).

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In F. R. Bach, & D. M. Blei (Eds.), *JMLR Workshop and Conference Proceedings*: *vol. 37*, *Proceedings of the 32nd international conference on machine learning, ICML 2015, Lille, France, 6-11 July 2015* (pp. 448–456). JMLR.org.

Jin, X., Xu, C., Feng, J., Wei, Y., Xiong, J., & Yan, S. (2016). Deep learning with S-shaped rectified linear activation units. In *Proceedings of the thirtieth AAAI conference on artificial intelligence* (pp. 1737–1743). AAAI Press.

Jou, C.-C. (1991). Fuzzy activation functions. In *[Proceedings] 1991 IEEE international joint conference on neural networks* (pp. 128–133). IEEE.

Karakose, M., & Akin, E. (2004). Type-2 fuzzy activation function for multilayer feedforward neural networks. In *2004 IEEE international conference on systems, man and cybernetics (IEEE Cat. No. 04CH37583), Vol. 4* (pp. 3762–3767). IEEE.

Karlik, B., & Olgac, A. V. (2011). Performance analysis of various activation functions in generalized mlp architectures of neural networks. *International Journal of Artificial Intelligence and Expert Systems*, *1*(4), 111–122.

Klabjan, D., & Harmon, M. (2019). Activation ensembles for deep neural networks. In *2019 IEEE international conference on big data (Big Data), Los Angeles, CA, USA, December 9-12, 2019* (pp. 206–214). IEEE.

Krizhevsky, A., & Hinton, G. (2009). Learning multiple layers of features from tiny images. In *Computer science department, University of Toronto, Tech. Rep, Vol. 1*.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2324.

Leshno, M., Lin, V. Y., Pinkus, A., & Schocken, S. (1993). Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural Networks*, *6*(6), 861–867.

Lin, M., Chen, Q., & Yan, S. (2013). Network in network. arXiv preprint arXiv:1312.4400.

Maas, A. L., Hannun, A. Y., & Ng, A. Y. (2013). Rectifier nonlinearities improve neural network acoustic models. In *Proc. Icml*.

Memisevic, R., Konda, K., & Krueger, D. (2014). Zero-bias autoencoders and the benefits of co-adapting features. *Stat*, *1050*(13).

Montalto, A., Tessitore, G., & Prevete, R. (2015). A linear approach for sparse coding by a two-layer neural network. *Neurocomputing*, *149*, 1315–1323.

Müller, B., Reinhardt, J., & Strickland, M. T. (2012). *Neural networks: an introduction*. Springer Science & Business Media.

Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted boltzmann machines, In *Proceedings of the 27th international conference on machine learning (ICML-10)* (pp. 807–814).

Netzer, Y., Wang, T., Coates, A., Bissacco, A., Wu, B., & Ng, A. Y. (2011). Reading digits in natural images with unsupervised feature learning.

Nguyen, A., & Korikov, A. M. (2017). Models of neural networks with fuzzy activation functions. In *IOP conference series: Materials science and engineering, Vol. 177*. IOP Publishing, Article 012031.

Nitta, T. (2013). Local minima in hierarchical structures of complex-valued neural networks. *Neural Networks*, *43*, 1–7.

Njikam, A. N. S., & Zhao, H. (2016). A novel activation function for multilayer feed-forward neural networks. *Applied Intelligence: The International Journal of Artificial Intelligence, Neural Networks, and Complex Problem-Solving Technologies*, *45*(1), 75–82.

Nwankpa, C., Ijomah, W., Gachagan, A., & Marshall, S. (2018). Activation functions: Comparison of trends in practice and research for deep learning. arXiv preprint arXiv:1811.03378.

Pang, Y., Sun, M., Jiang, X., & Li, X. (2017). Convolution in convolution for network in network. *IEEE Transactions on Neural Networks and Learning Systems*, *29*(5), 1587–1597.

Pedamonti, D. (2018). Comparison of non-linear activation functions for deep neural networks on mnist classification task. arXiv preprint arXiv:1804.02763.

Piazza, F., Uncini, A., & Zenobi, M. (1992). Artificial neural networks with adaptive polynomial activation function. In *Proc. of the IJCNN, Vol. 2* (pp. 343–349). Citeseer.

Piazza, F., Uncini, A., & Zenobi, M. Neural networks with digital LUT activation functions. In *Proceedings of 1993 international conference on neural networks (IJCNN-93-Nagoya, Japan), Vol. 2* (pp. 1401–1404).

Pinkus, A. (1999). Approximation theory of the mlp model in neural networks. *Acta Numerica*, *8*, 143–195.

Qian, S., Liu, H., Liu, C., Wu, S., & Wong, H.-S. (2018). Adaptive activation functions in convolutional neural networks.. *Neurocomputing*, *272*, 204–212.

Qiu, S., Xu, X., & Cai, B. (2018). Frelu: Flexible rectified linear units for improving convolutional neural networks. In *2018 24th international conference on pattern recognition (ICPR)* (pp. 1223–1228). IEEE.

Ramachandran, P., Zoph, B., & Le, Q. V. (2018). Searching for activation functions. In *6th international conference on learning representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, workshop track proceedings*.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., et al. (2015). Imagenet large scale visual recognition challenge. *International Journal of Computer Vision (IJCV)*, *115*(3), 211–252.

Scardapane, S., Scarpiniti, M., Comminiello, D., & Uncini, A. (2017). Learning activation functions from data using cubic spline interpolation. In *Italian workshop on neural nets* (pp. 73–83). Springer.

Scardapane, S., Van Vaerenbergh, S., Hussain, A., & Uncini, A. (2018). Complex-valued neural networks with nonparametric activation functions. *IEEE Transactions on Emerging Topics in Computational Intelligence*, *PP*.

Scardapane, S., Van Vaerenbergh, S., Totaro, S., & Uncini, A. (2019). Kafnets: Kernel-based non-parametric activation functions for neural networks. *Neural Networks*, *110*, 19–32.

Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R., & Lecun, Y. (2014). Overfeat: integrated recognition, localization and detection using convolutional networks. In *International conference on learning representations (ICLR2014), CBLS*.

Shah, A., Kadam, E., Shah, H., Shinde, S., & Shingade, S. (2016). Deep residual networks with exponential linear unit. In *Proceedings of the third international symposium on computer vision and the internet* (pp. 59–65).

Sibi, P., Jones, S. A., & Siddarth, P. (2013). Analysis of different activation functions using back propagation neural networks. *Journal of Theoretical and Applied Information Technology*, *47*(3), 1264–1268.

Simonyan, K., & Zisserman, A. (2015). Very deep convolutional networks for large-scale image recognition. In *3rd international conference on learning representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, conference track proceedings*.

Singh, Y., & Chandra, P. (2003). A class+ 1 sigmoidal activation functions for FFANNs. *Journal of Economic Dynamics and Control*, *28*(1), 183–187.

Smith, A. R. (1983). Spline tutorial notes. In *Computer graphics project, lucasfilm ltd (presented as tutorial notes at the 1983 and 1984 SIGGRAPH)*. Citeseer.

Sonoda, S., & Murata, N. (2017). Neural network with unbounded activation functions is universal approximator. *Applied and Computational Harmonic Analysis*, *43*(2), 233–268.

Springenberg, J., Dosovitskiy, A., Brox, T., & Riedmiller, M. (2015). Striving for simplicity: The all convolutional net. In *ICLR (Workshop Track)*.

Springenberg, J. T., & Riedmiller, M. (2014). Improving deep neural networks with probabilistic maxout units. *Stat*, *1050*, 19.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, *15*(1), 1929–1958.

Srivastava, R. K., Masci, J., Kazerounian, S., Gomez, F., & Schmidhuber, J. (2013). Compete to compute. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, & K. Q. Weinberger (Eds.), *Advances in neural information processing systems, Vol. 26* (pp. 2310–2318). Curran Associates, Inc..

Stevens, S. S. (1957). On the psychophysical law.. *Psychological Review*, *64*(3), 153.

Stinchcombe, M., & White, H. (1990). Approximating and learning unknown mappings using multilayer feedforward networks with bounded weights. In *Neural networks, 1990., 1990 IJCNN international joint conference on* (pp. 7–16). IEEE.

Sun, W., Su, F., & Wang, L. (2018). Improving deep neural networks with multilayer maxout networks and a novel initialization method. *Neurocomputing*, *278*, 34–40.

Sütfeld, L. R., Brieger, F., Finger, H., Füllhase, S., & Pipa, G. (2018). Adaptive blending units: Trainable activation functions for deep neural networks. arXiv preprint arXiv:1806.10064.

Szegedy, C., Ioffe, S., Vanhoucke, V., & Alemi, A. A. (2017). Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-first AAAI conference on artificial intelligence*.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., et al. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 1–9).

Tessitore, G., & Prevete, R. (2011). Designing structured sparse dictionaries for sparse representation modeling. In *Computer recognition systems 4* (pp. 157–166). Springer.

Trottier, L., Gigu, P., Chaib-draa, B., et al. (2017). Parametric exponential linear unit for deep convolutional neural networks. In *Machine learning and applications (ICMLA), 2017 16th IEEE international conference on* (pp. 207–214). IEEE.

Urban, S., Basalla, M., & van der Smagt, P. (2017). Gaussian process neurons learn stochastic activation functions. *Stat*, *1050*, 29.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., et al. (2017). Attention is all you need. In *Advances in neural information processing systems* (pp. 5998–6008).

Vecci, L., Piazza, F., & Uncini, A. (1998). Learning and approximation capabilities of adaptive spline activation function neural networks. *Neural Networks*, *11*(2), 259–270.

Wang, M., Liu, B., & Foroosh, H. (2018). Look-up table unit activation function for deep convolutional neural networks. In *2018 IEEE winter conference on applications of computer vision (WACV)* (pp. 1225–1233).

Wang, J., & Perez, L. (2017). The effectiveness of data augmentation in image classification using deep learning. *Convolutional Neural Networks Visualization Recognition*, 11.

Widrow, B., & Hoff, M. E. (1960). *Adaptive switching circuits*: *Tech. rep.*, Stanford Univ Ca Stanford Electronics Labs.

Widrow, B., & Lehr, M. A. (1990). 30 years of adaptive neural networks: perceptron, madaline, and backpropagation. *Proceedings of the IEEE*, *78*(9), 1415–1442.

Xu, B., Huang, R., & Li, M. (2016). Revise saturated activation functions. arXiv preprint arXiv:1602.05980.

Xu, B., Wang, N., Chen, T., & Li, M. (2015). Empirical evaluation of rectified activations in convolutional network. arXiv preprint arXiv:1505.00853.

Yamada, T., & Yabuta, T. (1992a). Neural network controller using autotuning method for nonlinear functions. *IEEE Transactions on Neural Networks*, *3*(4), 595–601.

Yamada, T., & Yabuta, T. (1992b). Remarks on a neural network controller which uses an auto-tuning method for nonlinear functions. In *[Proceedings 1992] IJCNN international joint conference on neural networks, Vol. 2* (pp. 775–780).

Zagoruyko, S., & Komodakis, N. (2016). Wide residual networks. In *Proceedings of the british machine vision conference 2016, BMVC 2016, York, UK, September 19-22, 2016*. BMVA Press.

Zoph, B., Vasudevan, V., Shlens, J., & Le, Q. V. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 8697–8710).