BIG DATA MINING AND ANALYTICS
ISSN 2096-0654 02/05 pp102–120
Volume 3, Number 2, June 2020
DOI: 10.26599/BDMA.2019.9020024

Feature Representations Using the Reflected Rectified Linear Unit (RReLU) Activation

Chaity Banerjee, Tathagata Mukherjee*, and Eduardo Pasiliao Jr.

Abstract: Deep Neural Networks (DNNs) have become the tool of choice for machine learning practitioners today. One important aspect of designing a neural network is the choice of the activation function to be used at the neurons of the different layers. In this work, we introduce a four-output activation function called the Reflected Rectified Linear Unit (RReLU) activation which considers both a feature and its negation during computation. Our activation function is "sparse", in that only two of the four possible outputs are active at a given time. We test our activation function on the standard MNIST and CIFAR-10 datasets, which are classification problems, as well as on a novel Computational Fluid Dynamics (CFD) dataset which is posed as a regression problem. On the baseline network for the MNIST dataset, having two hidden layers, our activation function improves the validation accuracy from 0.09 to 0.97 compared to the well-known ReLU activation. For the CIFAR-10 dataset, we use a deep baseline network that achieves 0.78 validation accuracy with 20 epochs but overfits the data. Using the RReLU activation, we can achieve the same accuracy without overfitting the data. For the CFD dataset, we show that the RReLU activation can reduce the number of epochs from 100 (using ReLU) to 10 while obtaining the same levels of performance.

Key words: deep learning; feature space; approximations; multi-output activations; Rectified Linear Unit (ReLU)

1 Introduction

Deep Neural Networks (DNNs)[1] have become the learning algorithm of choice for a lot of real-world large-scale machine learning tasks. They have been successfully used for solving problems in computer vision[2, 3], natural language processing[4], machine learning for radio frequency domains[5], robotics[6], and bioinformatics[7–10], to name a few areas of application. However, in spite of the success of deep neural networks, there are gaps in our understanding of how such deep learning machines work, considerable research and reshaping of our basic understanding about machine learning systems are required in order to plugin this gap[11]. Though the universal approximation theorem states that multi-layer feedforward networks are universal approximators[12], most of the real-world networks in use today do not satisfy the underlying assumptions of these theorems in one way or another. One important aspect of neural network design is the choice of the activation function to be used at different layers of the network. Activation functions are used to introduce non-linearity into the neural network computations and proper choice of activation functions is crucial for effective performance of neural networks. The early activation functions to gain traction were either the Sigmoid or the Tanh. Both are differentiable and have nice analytic properties that make them mathematically appealing. However, with the advent of deep neural architectures, practitioners realized that it was hard to train very deep neural networks with these activation functions as they are saturated activations.

• Chaity Banerjee is with Department of Idustrial & Systems Engineering, University of Central Florida, Orlando, FL 32816-2368, USA. E-mail: Chaity.BanerjeeMukherjee@ucf.edu.
• Tathagata Mukherjee is with the Department of Computer Science, University of Alabama in Huntsville, Huntsville, AL 35806, USA. E-mail: tathagata.mukherjee@uah.edu.
• Eduardo Pasiliao Jr. is with Air Force Research Labs, United States Air Force, Eglin Air Force Base, Shalimar, FL 32579, USA. E-mail: elpasiliao@gmail.com.
* To whom correspondence should be addressed.
Manuscript received: 2019-11-25; accepted: 2019-12-13

To get around this problem, the Rectified Linear Unit (ReLU) activation was introduced, which though non-differentiable at zero is non-saturated and hence speeds up training of deep neural models.

A function f is called non-saturated if and only if it satisfies the following:

$$|\lim_{x\to+\infty} f(x)| = +\infty \vee |\lim_{x\to-\infty} f(x)| = +\infty \quad (1)$$

Any function f that does not satisfy the above condition is saturated. Note that the idea of saturated and non-saturated functions is independent of their use in neural networks. Intuitively, a saturated function is one that "squeezes" the domain of the input. Thus for example, the Sigmoid activation function that is defined as $\sigma(x) = \frac{1}{1+\mathrm{e}^{-x}}$, is saturated as it maps any input to the range [0, 1], whereas the ReLU activation which is defined as $\sigma(x) = \max\{0, x\}$ is non-saturated as $\lim_{x\to+\infty}\sigma(x) = \infty$ and hence does not squeeze the input domain. Note that the ReLU activation is not differentiable at 0 and hence back propagation should theoretically fail at this point. However, in general, due to the precision issues of floating point numbers in computers, this situation seldom arises in reality and ReLU as defined above is known to work well with very deep neural networks[1]. Apart from ReLU and its single output variants like leaky-ReLU, there are multi-output variations of ReLU like Concatenated ReLU (CReLU). All of these ReLU variations are non-saturated and hence work well for training deep networks. Note that some recent works show that deep Tanh networks are able to converge with careful model initialization while deep Sigmoid networks still fail. We provide a detailed discussion of the state of the art in activation function in Section 2.

Apart from the property of saturation of the activation functions, another property that is important in the context of training deep networks is sparsity[13]. Sparse networks are important because they have less number of parameters and hence are easier to train and less prone to the problem of overfitting thus giving better generalization performance. Sparsity in deep networks is usually enforced through the use of regularizers, using the idea of dropout[1] which is another form of regularization or through the use of convolutional networks. However, activation functions can also be used for network sparsification[13]. Thus for example, the ReLU activation enforces sparsity by only considering the highly positive features and discarding the negative ones whereas variations like the CReLU enforce sparsity by considering either the highly positive or the highly negative features at a time. Though activation functions are limited in their ability to enforce network sparsity, coupled with the idea of saturation, they have a large impact on the performance of a deep neural network.

The idea of multi-output ReLU activation was first introduced independently in the context of convolutional networks in Refs. [14, 15]. Both papers introduced a form of two-output variation of the ReLU activation. In Ref. [14], the authors first examined existing convolutional neural network models and discussed an important property of the convolutional filters in the lower layers. More precisely, they observed that the filters in the lower layers formed pairs (i.e., filters with opposite phases). Inspired by this observation, the authors proposed a novel, yet simple and effective activation scheme called CReLU. CReLU is defined as follows: $\sigma(x) = (\max\{0, x\}, \max\{0, -x\})$, and hence is a two-output activation which doubles the depth of the activation from one layer to another. Thus, whenever x is positive, the activation is $(x, 0)$. But when x is negative, unlike the ReLU which discards the negative features, here the activation is $(0, -x)$. The intuition for this activation scheme is to allow a filter to be activated in both positive and negative directions while maintaining the same degree of non-saturated non-linearity. Note that though the authors motivated the use of this activation using the idea of capturing both the positive and negative phases, one may also look at this activation, when used in the context of fully connected networks, as the width of the output of the particular layer increasing. Thus, if the layer at which the activation is applied has n neurons, the resulting output will have $2n$ neural units. This in turn has the potential to improve the quality of approximations computed by the resulting network as wider networks can compute better approximations of the underlying function[12] for the same depth of the network[16].

Motivated by the efficiency of the two-output ReLU activation, in this paper, we introduce the idea of generalized non-saturated multi-output activations and study their performance on the task of multi-class classification. More precisely, we propose a generalized framework for a four-output ReLU activation. Our activation is non-saturated and enforces sparsity thereby guaranteeing the efficacy of the activation for training deep networks. As we consider a four-output activation, our function doubles the amount of non-saturated non-linearity and increases the depth of the activation by four. Note that this in turn has the potential to compute equivalent solutions as a deep network using a shallower variety for a given classification task. Unlike the concatenated ReLU where the activation considers the positive phase if the features are highly positive and the negative phase if the features are highly negative but not both at the same time, our activation considers both a feature and its negation at each neuron which automatically augment the data in the feature space, hence speeding up the learning process. We demonstrate the efficacy of the four-output activation for classification using the MNIST and CIFAR-10 datasets.

This paper is organized as follows: In Section 2, we present a detailed study of the various activation functions that have been studied in the context of neural networks; in Section 3, we formally introduce the four-output variation of the ReLU activation and study some of its properties. Finally, in Section 4, we describe the results of our experiments. We conclude the paper in Section 5.

2 Background and Related Work

Activation functions have been studied for a very long time, in fact since the early days of neural networks. Let us suppose that the input to a shallow (single hidden layer) neural network is given by the vector x. Let us suppose that the network is fully connected. Then given a single neuron of the hidden layer, the input to the neuron is given by $w^{\mathrm{T}}x$, where w is the vector of weights and it is learned through back propagation. If the single neuron does nothing but passes on this input to its output, the output of the neuron is also $w^{\mathrm{T}}x$, then the operation that is being performed at the neuron is nothing but a simple aggregation and if this happens at each of the nodes of the hidden layer, then the neural network becomes a system that simply outputs a linear combination of its inputs, which is not much of a learning. In order to get over this problem, we introduce the idea of an activation function at a node. The activation function, usually denoted by σ, takes the input to the node and adds "non-linearity" to it, thus the output of the node becomes $\sigma(w^{\mathrm{T}}x)$. Note that if σ is the identity mapping, then the output to a node is the linear combination of the input values. This is known as a linear activation and as we have seen above, it does not do much.

In order to introduce non-linearity in the neural network computation, the choice of $\sigma(\ )$ is of paramount importance. In the early years of neural networks, σ was usually chosen to be the Sigmoid activation which is defined as $\sigma(x) = \dfrac{1}{1+\mathrm{e}^{-x}}$. The advantages of Sigmoid activation functions are that they are easy to understand and use, but they are mostly used in shallow networks[17]. With the advent of deep learning, the use of Sigmoid activations has plummeted. Sigmoid activation shows some disadvantages, i.e., sharp damp gradients during back-propagation from deeper hidden layers to the input layers, gradient saturation, slow convergence, and non-zero centered output, thereby causing the gradient updates to propagate in a different direction. The unsuitability of the Sigmoid activation for training deep networks with random initialization was first studied in Ref. [18]. The authors noted that Sigmoid activation was "unsuited for deep networks with random initialization because of its mean value, which can drive especially the top hidden layer into saturation". Most of the problems with the Sigmoid activation can be explained by the fact that it introduces saturated non-linearity and hence this should be avoided for learning with deeper neural architectures.

Different variations of the Sigmoid activation have also been considered in the literature. For example, in Ref. [19], the authors introduced the hard Sigmoid activation. They considered a neural network with weights constrained to two values, namely, -1 and 1, and demonstrated the efficacy of the hard Sigmoid on three different standard datasets. The hard Sigmoid activation is defined as follows:

$$\sigma(x) = \max\left\{0, \min\left\{0, \frac{x+1}{2}\right\}\right\} \tag{2}$$

One of the advantages of hard Sigmoid in comparison to the soft Sigmoid as discussed above, is the fact that the hard Sigmoid activation has a lesser computation cost when implemented either in specialized hardware or software. Another variation of the Sigmoid activation has been used in reinforcement learning and is called the Sigmoid weighted Linear Units (SiLU)[20]. Finally, there is the variation of the Sigmoid activation called the Swish activation, which is defined as $\sigma(x) = x \cdot Sigmoid(x)$[21]. The Swish is a simple function and easy to implement and use, and does not suffer from the problem of vanishing gradients.

In order to overcome the issues with the Sigmoid activation, the hyperbolic tangent (Tanh) activation was introduced[22]. The Tanh activation is defined as

$$\sigma(x) = \frac{\mathrm{e}^{x} - \mathrm{e}^{-x}}{\mathrm{e}^{x} + \mathrm{e}^{-x}} \tag{3}$$

It should be noted that the Tanh activation is also a saturated activation function, because it squeezes the input into the range $[-1, 1]$. With the renewed interest in neural networks, the Tanh function becomes the preferred activation compared to the Sigmoid activation, because it leads to better training performance for multi-layer neural networks[23]. Variations of the Tanh activation have also been studied and successfully used in natural language processing[24]. However, the Tanh function does not solve the vanishing gradient problem of the Sigmoid activations. One of the main advantages of the Tanh activation is that it produces zero-centered output, thereby aiding the back-propagation process. A property of the Tanh activation is that it can attain a gradient of 1 only when the value of the input is $x = 0$. This makes the Tanh function produce some dead neurons during computation. A dead neuron is a condition where the activation weight is not used as a result of zero gradient. This limitation of the Tanh function has spurred further research into activation functions with the aim of resolving the problem. One of the products of this research is the invention of the ReLU activation function which we discuss in detail later.

Several activation functions have been inspired by the Tanh activation. The hard hyperbolic function defined as

$$\sigma(x) = \begin{cases} -1, & x < -1; \\ x, & -1 \leqslant x \leqslant 1; \\ 1, & x > 1 \end{cases} \tag{4}$$

has been used in natural language processing[24] and is a simpler and computationally less expensive version of Tanh. The Softmax activation is another variation inspired by the Tanh activation that is widely used. The Softmax activation is defined as

$$\sigma(x_i) = \frac{\mathrm{e}^{x_i}}{\sum_j \mathrm{e}^{x_j}} \tag{5}$$

and always produces values between 0 and 1. Furthermore, the output of the Softmax activation can be considered as probabilities as they always sum to 1. It must be pointed out that both Sigmoid and Softmax activations produce values between 0 and 1 and hence are saturated activations. While the Sigmoid activation is used for binary classification, and the Softmax activation is used for multi-class classification. Finally, there is the Softsign activation function which was first introduced in Ref. [25]. It is defined as

$$\sigma(x) = \frac{x}{|x| + 1} \tag{6}$$

The Softsign activation has been used in deep learning systems for regression and speech processing.

The ReLU activation was first proposed in Ref. [26]. As described earlier, the ReLU activation is defined as $\sigma(w^{\mathrm{T}}x) = \max\{0, w^{\mathrm{T}}x\}$, where x is the input and w is the parameter learned using back-propagation. ReLU has been the most widely used as well as the most successful activation function till date and has been used to solve a wide variety of problems[21]. The ReLU activation provides better performance both in terms of training time and generalization capability when compared with the Sigmoid or the Tanh activations[27]. Furthermore, ReLU is a linear function and hence preserves the linear properties of the underlying features and is easy to optimize using gradient descent. Another nice property of ReLU is that it introduces sparsity in the hidden units as it squeezes the input values from zero to infinity (in practice the upper bound is the maximum possible value of the feature).

In spite of its successes, ReLU is not without drawbacks. ReLU has a limitation that it easily overfits the data compared to the Sigmoid activation. The idea of "dropouts"[1] has been used to reduce the chances of overfitting with ReLU and several variations of ReLU activation have been proposed and studied in the community with the aim of alleviating this problem. In recent time, there has been a concerted effort to understand why deep networks work the way they do[16] and as a part of this effort, researchers are also trying to understand why the ReLU activation outperforms most of the classical activation functions by considering the approximation power of ReLU networks. For example, in Ref. [28], the authors investigated the family of functions represented by DNNs using ReLU activations. There are also some work towards generalizing the ReLU activation. For example, in Ref. [29], the authors studied the problem of learning a generalized ReLU which has the form of $\max(0, w^{\mathrm{T}}x)$, reliably in polynomial time whereas in Ref. [30], the authors studied generalizations of the two-output concatenated ReLU empirically.

One of the most commonly used variations of ReLU is the "leaky-ReLU', which was first proposed in Ref. [31] and used in the contest of natural language processing. The leaky-ReLU is similar to the ReLU but instead of completely ignoring the negative features, as is done in ReLU, it maintains a small slope in the activation function for the negative features, thus considering a fraction of the negative features for the feature space computation. The leaky-ReLU activation is defined as

$$\sigma(x) = \begin{cases} x, & x > 0; \\ \alpha x, & x \leqslant 0 \end{cases} \tag{7}$$

where the parameter α which is usually chosen to be a small positive number, was introduced in order to get around the problem of dead neurons that is encountered for the ReLU activation. Due to this parameter, the gradients are never equal to zero during the training time which in turn solve the dead neuron problem. It must be pointed out that the leaky-ReLU does not lead to a significant improvement in the results as compared to the ReLU. The only gain from using the leaky-ReLU activation is the fact that the gradients are non-negative throughout the training process and there is some change in the sparsity of the underlying features that are computed.

The parameter α for the leaky-ReLU is chosen to be a small constant. However, there is the possibility of "learning" this parameter using back propagation during the learning process and the Parametric ReLU (PReLU) activation achieves exactly that. PReLU is defined in Ref. [7], but the parameter α is learned during the training phase. It was reported by the authors that PReLU was better than ReLU for large-scale image recognition tasks and indeed networks using the PReLU surpassed humans for the task of classification using the imagenet dataset[32]. Another variation of the leaky-ReLU which is similar is called the randomized leaky ReLU. The activation function is defined in the same way as in Ref. [7], but the parameter α is neither a constant nor is learned. Instead, it is chosen randomly from a subset of [0, 1]. Another variation of ReLU that uses learnable parameters is called the S-Shaped ReLU. Here instead of the scale parameters being learnable, the thresholds are also learned from the data. Interested readers can refer to Ref. [33] for details on this activation.

One interesting variation of the ReLU activation that has been studied is the Flexible Rectified Linear Unit (FReLU)[34]. ReLU networks miss the benefits from negative values and hence in this paper, the authors proposed a novel activation function to further explore the effects of negative values. By redesigning the rectification point of ReLU as a learnable parameter, FReLU expands the states of the activation output. When the network is successfully trained, FReLU tends to converge to a negative value, which improves the expressiveness and thus the performance. Furthermore, FReLU is designed to be simple and effective without exponential functions to maintain low-cost computation.

A smoothed version of ReLU called the Softplus activation function was introduced in Ref. [35]. It is defined as $\sigma(x) = \log(1 + \mathrm{e}^x)$. This function has smoothing and non-zero gradient properties which enhance the stabilization and performance of deep neural networks designed with Softplus units. The Softplus activation has been used for statistical applications and speech recognition, and in general is known to converge faster than ReLU and Sigmoid activations.

Most of the activation functions inspired by ReLU, that we have studied till now, extend ReLU by considering scaled versions of the negative features, where the scaling factor is either a fixed constant or learned from the data. The Exponential Linear Unit (ELU) is also inspired by ReLU but instead of considering scaled versions of the negative features, it considers scaled versions of the exponentiation of these features. Thus the ELU is defined as

$$\sigma(x) = \begin{cases} x, & x > 0; \\ \alpha(\mathrm{e}^x - 1), & x \leqslant 0 \end{cases} \tag{8}$$

where α is an ELU hyper parameter that controls the saturation point for negative inputs, which is usually set to 1. The advantages of ELU are in that it achieves faster training time and generalization ability, specially with networks that have five or more layers and as ELU has a clear saturation plateau for the negative features, it thereby learns more robust feature representations. However, ELU has its own problem, namely, it does not center the values at zero. In order to address this problem, the parametric ELU activation was proposed, which is defined as

$$\sigma(x) = \begin{cases} cx, & x > 0; \\ \alpha(\mathrm{e}^{x/b} - 1), & x \leqslant 0 \end{cases} \tag{9}$$

where α, b, and $c > 0$ are parameters. Here c changes the slope in the positive quadrant, b controls the scale of the exponential decay, and α controls the saturation in the negative quadrant. Interested readers can consult Ref. [36] for further details. Another variation of the ELU is the Shifted Exponential Linear Units (ShELU)[37]. Here the authors applied a δ shift to the input values near the origin and showed that this leads to improved learning using convolutional networks.

One of the most interesting variations of ReLU-like activations is the idea of the Maxout layer that was first introduced by Goodfellow et al.[38] in 2013. In a typical neural network, given the input $x \in \mathbb{R}^d$ and a hidden layer having m nodes, the output of the hidden layer is computed as $z \in \mathbb{R}^m$ where $z_i = \sigma(w_i^{\mathrm{T}} x + b_i)$, and w_i and b_i are learned parameters. However, for the

Maxout network with the same settings, corresponding to one hidden layer node, k pseudo-nodes are created. Thus for the i-th hidden layer node, it creates k nodes $z_{i1}, z_{i2}, \ldots, z_{ik}$ where $z_{ij} = w_{:ij}^{\mathrm{T}} x + b_{ij}$, and $w_{:ij}$ and b_{ij} are learned parameters. Note that we use the ":" in $w_{:ij}$ in order to emphasize the fact that the weights are corresponding to the "pseudo-nodes" that correspond to the i-th hidden layer node. Finally, the output of the i-th hidden layer node is obtained as $h_i(x) = \max_j z_{ij}$, $i = 1, 2, \ldots, m$. Thus intuitively, the output of each hidden layer node is a line and hence the m linear outputs together can be thought of as a piece-wise linear curve. In fact, the authors showed that the Maxout networks are universal approximators that use piece-wise linear functions to approximate any function. Also note that for the Maxout activation, the entire hidden layer can be thought of as the activation function and hence we can say that the Maxout layer is a multi-output activation function.

All the activation functions that we have discussed till now are single-output activations. They take a single value as the input and return a single value as the output. However, there are variations of activation functions that increase the depth of the activation by having more than one output and these are called multi-output activations. Two-output activations were introduced simultaneously in Refs. [14,15]. They used similar idea but the authors in Ref. [14] did more experiments and also proved a reconstruction property of the corresponding activation when used in context of convolutional networks. The CReLU is defined as follows: $\sigma(x) = (\max\{0, x\}, \max\{0, -x\})$. Note that this is a two-output function: The output of the function is either $(x, 0)$ or $(0, -x)$ depending on the value of the input x. Furthermore, this is not equivalent to the absolute value function which is a single-output function. The concatenated ReLU maintains the same degree to non-saturated non-linearity as the ReLU activation while at the same time doubling the depth of the activation. CReLU can be interpreted in two ways, the first considers the phases of the input feature being learned by the system, while the other, and not so common interpretation is that it makes the output layer wider, thereby increasing the approximating power of the resulting network[16]. Generalizations of this activation was empirically studied in Ref. [30], where the authors considered different scaling factors for the input features. They showed that by parameterizing the concatenated ReLU, which they called the generalized ReLU activation, there is room for improving the feature space learned by the deep network. Though they did not consider the problem of automatically learning the scaling parameter, it is possible to do so during the training phase using back propagation.

We have seen before that the concatenated ReLU activation accounts for both the positive phase as well as the negative phase, depending on the value of the input x. For the generalization of the ReLU, the authors in Ref. [30] considered not only the positive as well as the negative phases (that is x and $-x$), but also the effect of different choices of a phase parameter, which can be different depending upon whether x is positive or negative. They also considered the effect of shifting the point of discontinuity at the same time as the change of phase is done. Taking all these into account, the authors defined the generalized ReLU activation as

$$\sigma(x) = (\alpha \cdot \max\{0, \psi(x)\} + \gamma, \beta \cdot \max\{0, \phi(-x)\} + \delta) \tag{10}$$

where α, β, γ, and δ are hyper-parameters to be either learned from the data or selected empirically. $\psi()$ and $\phi()$ are functions of the input x. The authors selected the phase parameters as shown in Fig. 1 and also selected ψ and ϕ to be identity mappings. In Section 3.1, we introduce the four-output reflected ReLU which has been inspired by the success of these "two-output" variants.

Activation functions have also been studied from the perspective of the quality of the approximations that they compute as well as the types of feature spaces that are generated by them. For example, in Refs. [16,39], the authors studied the power of deeper networks with continuously differentiable activation functions for the

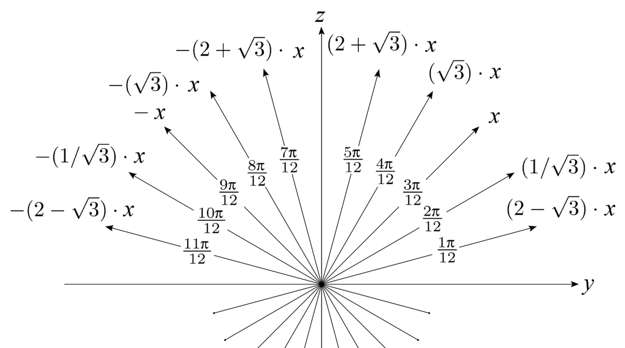

Fig. 1 Choice of the phase parameters.

task of learning discriminative features. Similarly in Ref. [40], the authors studied how the ReLU activation computed piece-wise linear approximations of functions. With the same goal in mind, namely, in order to understand why deep networks work, researchers have also studied such networks from the perspective of the features that they compute. For example, in Ref. [41], the authors studied the feature representations computed by autoencoders from a theoretical perspective and then studied the problem of input feature reconstruction from the computed features. Another direction of research in deep neural networks has been in relation to designing new and more efficient deep architectures for efficient training and testing for general classes of problems[42]. Several different deep architectures are available, such as GoogleNet and AlexNet[43], that can be used as an "off-the-shelf" deep network for many different tasks. These can also be modified and used with other novel architectures for intermediate feature computation[44]. In this paper, we do not study the activation function being introduced from a theoretical perspective, neither do we build novel architectures using the newly introduced activation. Our goal in this work is to show that the Reflected ReLU (RReLU) activation which is a non-saturated sparse function that produces simultaneous activations, and can be used in any neural network for computing possibly better models for both the tasks of classification and regression. Next we introduce the RReLU activation function.

3 Reflected ReLU Activation

Let the features of the i-th training example of the neural network be denoted by $x^i = (x_1^i, x_2^i, \dots, x_n^i)$ where $i = 1, 2, \dots, m$. Let us assume that the first hidden layer has η neurons where each neuron uses the activation function $\sigma()$. For the ease of description, let us also assume that the network is fully connected from the input layer to the hidden layer. Let the weight of the edge between input feature k and hidden unit l be denoted by w_{kl}. Then the output of the l-th hidden unit is given by $\sigma(\sum_{k=1}^{n} w_{kl} x_k + b_l)$ where b_l is the bias for this hidden unit. CReLU is the standard feed forward neural network model where the parameters are the weights and biases that are learned from the training data using back propagation. As seen from the expression of the output, the activation function σ plays an important role in adding non-linearity to the output of the network. Thus if we choose σ to be the linear activation, then the output of the hidden layer neuron is given by $\sum_{k=1}^{n} w_{kl} x_k + b_l$ which is nothing but a linear combination of the inputs. If this is the case with each node (neuron), then the output of the entire network is a linear combination of its inputs and hence the network is of limited utility as it can only estimate functions given by linear combinations of the input. Note that the activation function σ introduces non-linearity into the computations of the neural network but ideally for training deeper network architectures, we want to use non-saturated non-linearity which also increases the depth of the activation function. With this aim in mind, we are now ready to describe the reflected ReLU activation. This is inspired by the idea of concatenated ReLU, however instead of having a single active non-zero output at a time as is the case with concatenated ReLU, and the RReLU activation always has two active outputs among four possible outputs.

3.1 Reflected ReLU: Definition

As mentioned before, the ReLU activation considers only the positive features and discards the negative features by setting them to zero. In case of a fully connected network, this is done so as to consider only highly discriminative features which are defined as those that result in a positive response under the ReLU activation. In case of convolutional neural networks, ReLU-based filtering is supposed to facilitate the exploitation of discriminative information by de-noising the filter detections (negative values being filtered out as the noise). In the context of convolutional neural networks, ReLU and pooling filter out a lot of information from their input. The concatenated ReLU activation function modifies the convolutional block in order to keep more information after application of the ReLU. The idea of concatenated ReLU and its variants are motivated by the observation that information from the strong negative detections is as important as the strong positive detections, but this information is totally left out by the ReLU activation. For fully connected networks, the concatenated ReLU considers both the strongly positive features as well as the strongly negative ones.

The reflected ReLU activation borrows these ideas from the concatenated ReLU. However, the concatenated ReLU and its variations still consider either the strongly positive responses (features) or the strongly negative responses but do not consider simultaneously the strongly positive responses and their negation (or the strongly negative responses and their negation). Note that in two dimensions, given a feature x, its negation is simply its reflection in the x-axis. The reflected ReLU activation considers an input x and its negation simultaneously and is defined as follows:

$$\sigma(x) = (\max\{0,x\}, \max\{0,-x\}, \min\{0,x\}, \min\{0,-x\}).$$

Note that when $x \geqslant 0$, this translates to $\sigma(x) = (x, 0, 0, -x)$, whereas when $x < 0$, this translates to $\sigma(x) = (0, -x, 0, x)$. Furthermore, the activation function is sparse as two of the four possible outputs are zero for any given input x. We also note that this activation is non-saturated but unlike the concatenated ReLU, it doubles the degree of unsaturation, and the number of neurons in the output layer is quadrupled by the reflected ReLU, thus automatically increasing the width of the network which in turn may help in computing better approximations of the target functions.

In the context of a Convolutional Neural Network (CNN), the reflected ReLU can be interpreted as a simultaneous min-max activation (see Fig. 2). Thus given a convolutional filter, we take the output of the filter and concatenate it with the negative of the output. This concatenated output is now passed through two activations, i.e., the first one being $\max\{0, x\}$ which is nothing but the ReLU and the second is $\min\{0, x\}$, where x denotes the concatenation of the output of the convolutional filter and its negation.

Variations of reflected ReLU. The definition of RReLU as we have used above is amenable to several variations through the change of slope and origin as was done for the generalized ReLU in Ref. [30]. For example, for a slope-based variation, one can define the reflected ReLU as follows:

$$\begin{aligned}\sigma(x) = (&\alpha \cdot \max\{0, x\}, \alpha \cdot \max\{0, -x\},\\ &\beta \cdot \min\{0, x\}, \beta \cdot \min\{0, -x\}),\end{aligned}$$

where α and β are slope parameters that can in general be different or the same and may also be learned in a data driven paradigm. As discussed in Ref. [30], one can also choose them empirically based on the choices

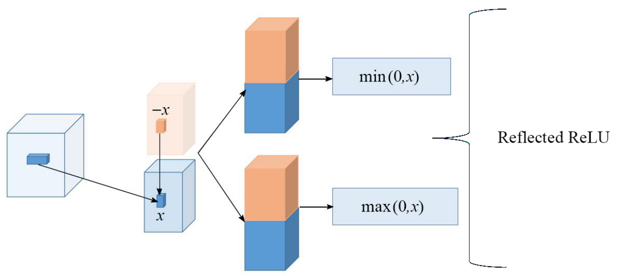

Fig. 2 Reflected ReLU: Simultaneous min-max convolutions.

as shown in Fig. 1. Note that with this variation, one can control the fraction of the positive and negative phases of the features that will be used for the neural network training. We also present results for a fixed choice of these phase parameters in the experimental section. In order to distinguish this variation from the definition of the RReLU as introduced before, we call this variation the parameterized RReLU and the version defined earlier the simple RReLU. Responses obtained from both variations are shown in Figs. 3 and 4.

3.2 Reconstruction property

As noted in Ref. [14], a notable property of concatenated ReLU is information preservation: Concatenated ReLU preserves both the negative and the positive linear responses after convolution (thus if the response is positive, then it is preserved through $\max\{0, x\}$; while if the response is negative, then it is preserved through $\max\{0, -x\}$). A direct consequence of information preservation is the ability to reconstruct the input to the convolutional layers, which are given the output of CReLU. Reconstruction property of a CNN implies that the features it computes are representative of the input data and this aspect of convolutional neural networks is important in order to understand the inner working of deep convolutional networks[14, 45]. Note that in order to be able to reconstruct the output of the convolutional

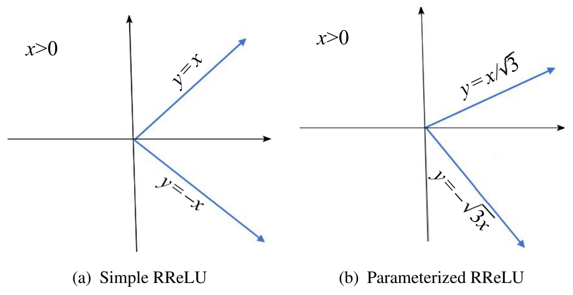

Fig. 3 Positive response plot for RReLU. The x-axis is the input feature and y-axis is the response corresponding to input feature x.

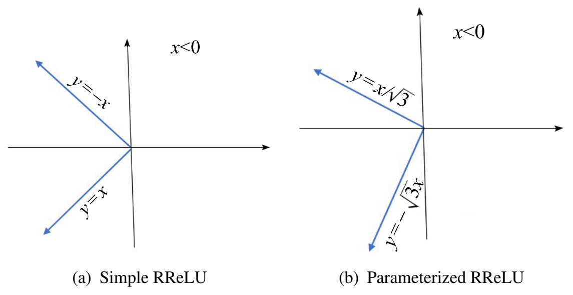

Fig. 4 Negative response plot for RReLU. The x-axis is the input feature and y-axis is the response corresponding to input feature x.

layer after application of the activation function, the activation should preserve the features computed by the convolutional layers and hence the reconstruction property should apply to the output of the activation function. Intuitively, if an activation function satisfies the reconstruction property, then it is "lossless" and hence conducive to computation of the best possible features using the output of the convolutional layers. Next, we show that the reflected ReLU activation satisfies the reconstruction property. The proof is simple and follows from a similar result for the concatenated ReLU as stated in Ref. [14]. We state the result in the context of reconstruction from the features computed using a single convolutional layer without max-pooling. The result for the case when max-pooling is used is similar and we omit it here as it also follows through the application of a similar result stated in Ref. [14].

Theorem 1. Let $x \in \mathbb{R}^n$ be the input vector and let w be the $d \times k$ weight matrix such that the columns of the matrix correspond to k convolutional filters. Let $x = x' + (x - x')$ where $x' \in range(w)$ and $(x - x') \in ker(w)$. Then we can reconstruct x' with $f_{CNN}(x) = RReLU(w^{\mathrm{T}}x)$.

Proof. The proof of the result follows using the reconstruction algorithm as given in Ref. [14] (replicated as Algorithm 1). Note that the RReLU is defined as

$$\sigma(x) = (\max\{0, x\}, \max\{0, -x\}, \min\{0, x\}, \min\{0, -x\}).$$

The first part of the activation $(\max\{0, x\}, \max\{0, -x\})$ is nothing but the CReLU and hence by Algorithm 1, this satisfies the reconstruction property. Now referring back to Fig. 2, we see that when applied to a single convolutional layer, RReLU consists of two parts, of which the first part is nothing but CReLU which is concatenated with its negation to get the RReLU activation. Hence in order to reconstruct the input from the output of RReLU, we simply apply Algorithm 1 to the output of the first part of the RReLU activation

Algorithm 1 Reconstruction over simgle convolution without max-pooling[14]

```
1: function RECONSTRUCT
2:     $f_{CNN}(x) \leftarrow$ convolutional features
3:     $w \leftarrow$ weight matrix
4:     $z \leftarrow RReLU^{+^{-1}} f_{CNN}(x)$
5:     $(w^{\mathrm{T}})^{+} \leftarrow$ Moore-Pensore-Inverse $(w^{\mathrm{T}})$
6:     Reconstruction $X \leftarrow (w^{\mathrm{T}})^{+} z$
7:     return $X$
8: end function
```

thereby generating the input. This completes the proof. ■

We are now ready to discuss the results using the reflected ReLU activation for the task of classification with the MNIST and CIFAR-10 datasets.

4 Experimental Setup and Result

In order to test the efficacy of the reflected ReLU activation, we used it for the task of classification on the MNIST and CIFAR-10 datasets and regression on a novel Computational Fluid Dynamics (CFD) dataset. We used a baseline neural network using the ReLU activation for the experiments and compared the results of classification (and regression) from this network with those obtained from the same network using the reflected ReLU activation instead of ReLU. We used the validation (or test) accuracy as a measure of how well the network performed with the given activation. We started our discussion by describing the MNIST dataset and the baseline network that we used for the same. Then we stated the results of our experiments with the MNIST data. This is followed by a similar discussion about the CIFAR-10 dataset and the CFD dataset. All the experiments were run on a computer having a 16 core AMD Threadripper CPU, 128 GB of RAM, a GTX 2080 Ti graphics card, running Ubuntu 18.04, and NVIDIA propriety drivers.

Choice of activation function. For our experiments, we decided to use two variations of the reflected ReLU activation, namely, the simple reflected ReLU and the parameterized Reflected ReLU (pRReLU). For the parameterized version, we used the parameters $\alpha = \frac{1}{\sqrt{3}}$ and $\beta = -\sqrt{3}$. Our choice of the parameters is not based on any optimization criteria but is done in a way as to make the positive and negative responses asymmetric (since in the case of simple RReLU, the responses are symmetric). Since the reflected ReLU is a form of data augmentation technique in the feature space (see discussion in Section 4.2), choosing the responses in a way that they are asymmetric has the possibility of making the resulting augmentation technique robust to artifacts in the feature space that may introduce bias. As mentioned in Ref. [30], the parameters for any parameterized version of activation function should ideally be chosen by optimizing the resulting loss. However, since the goal of this paper is to introduce a new activation function and not to study parameterized activations in general, we have refrained from taking that route, since we feel that our choice of parameters, though empirical, is enough to establish our thesis for this paper and shows the efficacy of the reflected ReLU activation.

4.1 Result for MNIST data

The MNIST dataset consists of 60 000 gray-scale images of handwritten digits (the numbers 0 – 9), each image having dimension 28×28. We used 50 000 images for training and 10 000 images for testing. The MNIST dataset is known to be an easy dataset compared to the other standard datasets being used with deep learning systems. As a result, we decided to use a shallow (having less than 3 hidden layers) fully connected network to keep the resulting model simple. Note that simple convolutional networks can obtain pretty high accuracy with the MNIST dataset[15] and so our expectation was that a fully connected network would get comparable results albeit may be a little less in terms of the accuracy.

Our network has input size of 784 (corresponding to the vectorized versions of 28×28 images). This is followed by two hidden layers each having 256 neurons. The output of the first hidden layer is passed through an ReLU activation and this is the input of the second hidden layer which simply computes a linear combination of its inputs (that uses just a linear activation). The output layer has 10 neurons corresponding to the 10 classes of the MNIST data. Note that this is a very simple network that uses a non-saturated non-linearity in one layer only. As mentioned before, we decided to keep the network simple in order for ease of comparison with the results from using the reflected ReLU. We used the Adaptive Moment Estimation (ADAM)[46] optimizer to minimize a softmax cross entropy loss. We trained the network with a batch size of 128 for 5000 steps. We trained the network several times and recorded the average training and testing errors across different runs. To our surprise, the average training accuracy with ReLU hovered 0.11–0.14 while the average test accuracy hovered between 0.09–0.11. The results for one of the trained models is shown in Table 1 and the plots of the training & testing accuracy for different mini-batch trainings are shown in Fig. 5.

Table 1 also shows the results of our experiments with the same network architecture when the ReLU activation is changed to the reflected ReLU with phase parameters $\alpha = \frac{1}{\sqrt{3}}$ and $\beta = -\sqrt{3}$. Figure 6 shows the plot of the training & testing accuracy for training over several mini-batches under the same settings. Finally, Fig. 7 shows the results for the simple RReLU under the same settings. Note that we do not change anything else in the network except the activation function applied to the first layer, which is changed from ReLU to parameterized/simple RReLU. As seen from Table 1 and Fig. 6, with this simple change, the training accuracy jumps from 0.110 to 0.984 and the testing accuracy jumps from a meager 0.09 to around 0.97, more than 90% increase from the baseline results with ReLU, under both

Table 1 Results for MNIST network with ReLU, pRReLU, and simple RReLU.

Activation	Train-Acc	Test-Acc
ReLU	0.110	0.09
pRReLU	0.984	0.97
Simple RReLU	0.960	0.96

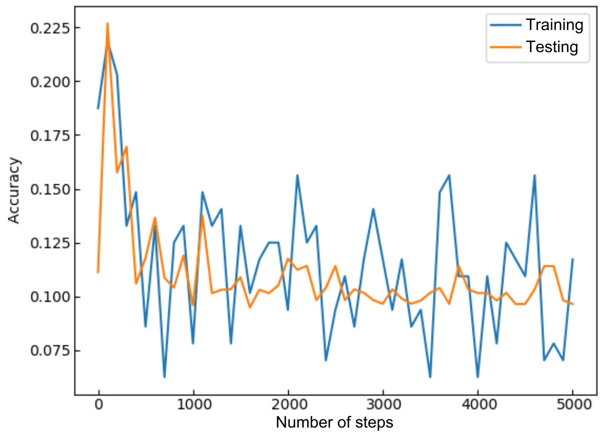

Fig. 5 Training & testing accuracy of baseline MNIST network (ReLU).

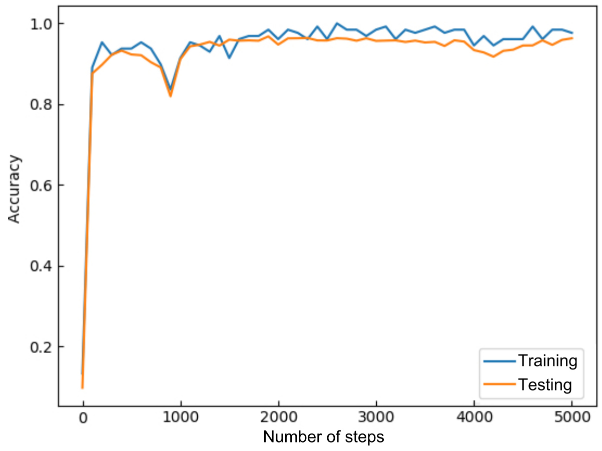

Fig. 6 Training & testing accuracy of MNIST network using RReLU with phase parameters $\alpha = \frac{1}{\sqrt{3}}$ and $\beta = -\sqrt{3}$.

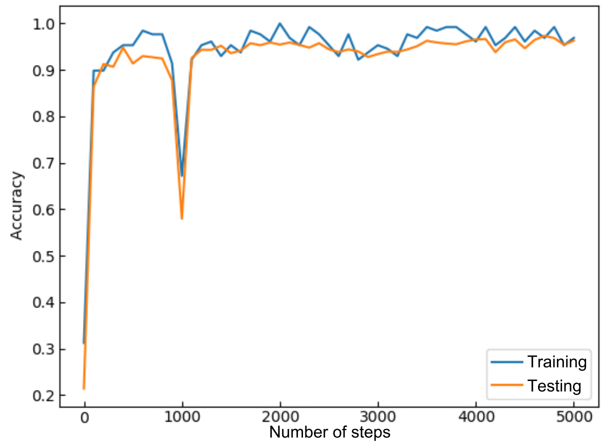

Fig. 7 Training & testing accuracy of MNIST network using simple RReLU.

the parameterized as well as the simple RReLU. That it was possible to increase the accuracy of the network by over 90% goes on to show the efficacy of the reflected ReLU activation.

Next, we present the results of our experiments with the CIFAR-10 image classification dataset. For the MNIST experiments, we had intentionally chosen a simple shallow network that was fully connected. Though our network was not optimized the reflected ReLU activation that was managed to get near state-of-the-art accuracy with the dataset. Now we wanted to test the activation with a more complicated network. The CIFAR-10 dataset is known to be hard for various reasons and fully connected networks are known to perform badly with the data. As a result, we started with a deeper convolutional network and optimized it with the ReLU activation. We used the results obtained from this network as the baseline for our comparison.

4.2 Result for CIFAR-10 dataset

The CIFAR-10 small image classification dataset[47] consists of 50 000, 32×32 color training images and 10 000 test images. Each image belongs to one of 10 classes: airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. Each class contains 6000 images and there is a wide variation within the images in each class. For example, the images for the class “birds” contain birds of several different types, each image containing birds at different magnifications. Furthermore, there are only 6000 images in each class of which around 5000 are used for training. This is too small a set to capture all the variability in the data. This makes it hard to design moderately deep convolutional networks that perform relatively well (testing accuracy over 0.7) for this dataset. We ran the experiments on a system having a 16-core AMD Threadripper CPU, 128 GB of RAM, running Ubuntu 16.04, and GTX 2080 Ti graphics card with 11 GB of graphics memory.

Our baseline CIFAR-10 network is convolutional in nature having a total of seven hidden layers, one input layer and one output layer of size 10. The first hidden layer is a convolutional layer and uses 32 filters each of size 3×3. The output of this layer is passed through the ReLU activation and a second convolutional layer having 64 filters each of size 3×3. The output from this is again passed through an ReLU activation followed by a max-pooling layer with a pool size of 2×2. A dropout with probability of 0.25 is applied after the max-pool. This is followed by two more convolutional layers followed by the ReLU activation where the first convolutional layer applied 128 filters and the second used 256 filters, each filter being of size 3×3. These are followed by a max-pooling layer and dropout as before. After the convolutional layers, we flatten the output and pass it through a fully connected layer having 512 neurons, an ReLU activation, and a dropout with probability of 0.5. Finally, we have another fully connected layer with 10 neurons as the final output layer. Note that for the baseline network as described above, all the activation functions were ReLU. The results of using the baseline network for the CIFAR-10 classification is shown in Table 2 for training with 20 epochs. Note that the network overfits with ReLU.

Figure 8 show the results for the same network architecture but with all the activation functions for the convolutional layers changed to parameterized RReLU (denoted by pRReLU in Table 2) with phase parameters $\alpha = \dfrac{1}{\sqrt{3}}$ and $\beta = -\sqrt{3}$. The fully connected layer still uses the ReLU activation and we train for 20 epochs. Figure 8 shows how the accuracy varies over the different epochs. We note that there is negligible decrease in the testing performance compared to the baseline network but there is no overfitting. Note that we obtained similar results when the simple RReLU activation was used instead of the parameterized version

Table 2 Results for CIFAR-10 network with ReLU, pRReLU (single layer), pRReLU, and simple RReLU.

Activation	Train-Acc	Test-Acc
ReLU	0.8161	0.7838
pRReLU (single layer)	0.8387	0.8004
pRReLU	0.7483	0.7805
Simple RReLU	0.7445	0.7693

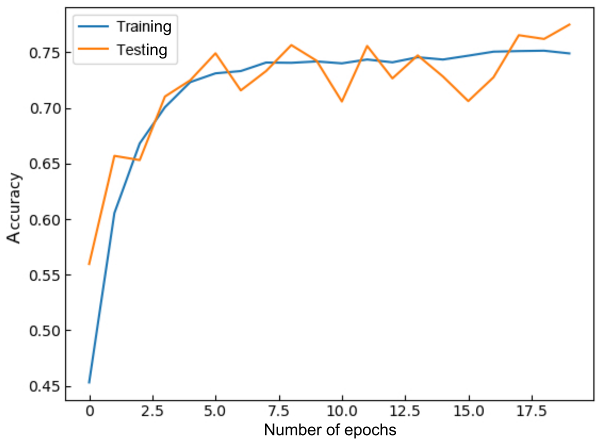

Fig. 8 Training & testing accuracy of CIFAR-10 network with RReLU and phase parameters $\alpha = \frac{1}{\sqrt{3}}$ and $\beta = -\sqrt{3}$ in all convolutional layers. Note that there is no overfitting.

of the function (see Table 2), that is as in the case of parameterized RReLU, there was no overfitting. It must also be pointed out that using any variation of the RReLU activation function in all the four convolutional layers quadruples the total number of parameters at each layer compared to the ReLU activation. However, the network still does not overfit, as it does when using the ReLU activation in all the layers. This observation is counter-intuitive and is most likely due to the fact that the RReLU activation can be interpreted as an automatic data augmentation technique in the feature space where every feature is augmented with its negation. This results in implicitly increasing size of the training set which in turn compensates for the increase in the number of parameters.

In order to test this hypothesis, we used the parameterized RReLU activation in the first or second convolutional layer only, using the ReLU activation in the remaining layers of the network. Note that if our hypothesis is wrong and RReLU reduces overfitting simply by the virtue of the quality of the features computed, then using RReLU in one of the earlier layers of the overfitting network (that uses ReLU in all the other layers) would reduce the overfitting. This is because a single application of RReLU would lead to computation of higher quality features which in turn would lead to an efficient model that will generalize well. On the other hand, if this is not the case, then we can safely conclude that the efficiency of RReLU is not simply due to the quality of the features computed, which in turn would support our assertion that RReLU is efficient and eliminates overfitting due to automatic data augmentation in the feature space. Table 2 also shows the results for the CIFAR-10 network architecture when the parameterized RReLU activation function is used only after the first convolutional layer. We note that though there is an improvement in the testing accuracy, the problem of overfitting remains. Figure 9 shows the plot of the training & testing accuracy over the different numbers of epochs under this setting. Similarly, Fig. 10 shows the plot of the training & testing accuracy over the different numbers of epochs when the parameterized RReLU activation is applied only after the second convolutional layer. The problem of overfitting is present in this case as well, which leads us to conclude that indeed the efficacy of RReLU activation is due to the fact that it works as an automatic data augmentation

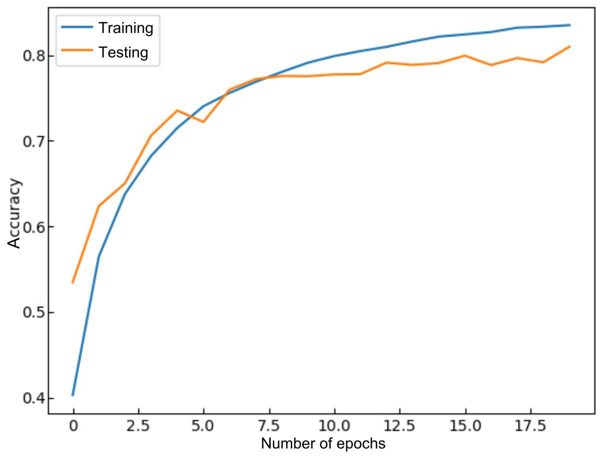

Fig. 9 Training & testing accuracy of CIFAR-10 network with RReLU and phase parameters $\alpha = \frac{1}{\sqrt{3}}$ and $\beta = -\sqrt{3}$ after the first convolutional layer only (model overfits).

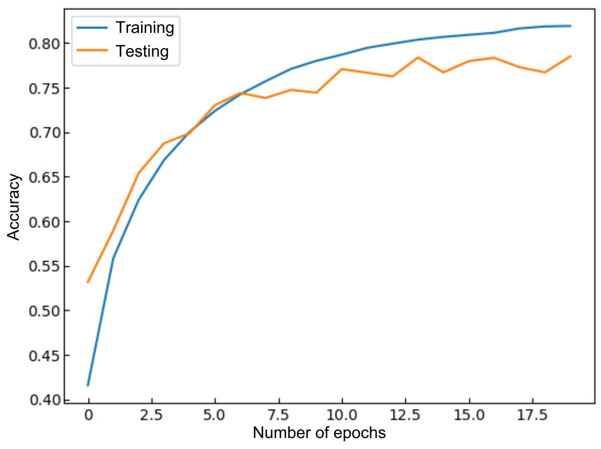

Fig. 10 Training & testing accuracy of CIFAR-10 network with RReLU and phase parameters $\alpha = \frac{1}{\sqrt{3}}$ and $\beta = -\sqrt{3}$ after the second convolutional layer only (model overfits).

technique in the feature space, automatically considering the negation of each of the computed features, which in turn offsets the effect of increasing the number of parameters and helps to learn a more robust model from the data.

In order to understand the computation being performed by the convolutional network with the RReLU activation, we ran the training several times, each time for a different number of epoch. In each case, we looked at one test image and the corresponding filters and feature maps learned by the network after the second convolutional layer. The test image that was used is shown in Fig. 11.

Next, we consider the feature maps learned by the network after full training with 20 epochs using both the ReLU and RReLU activations. Figures 12 and 13 show the feature maps learned by the network after 20 epochs in the second convolutional layer. Note that these

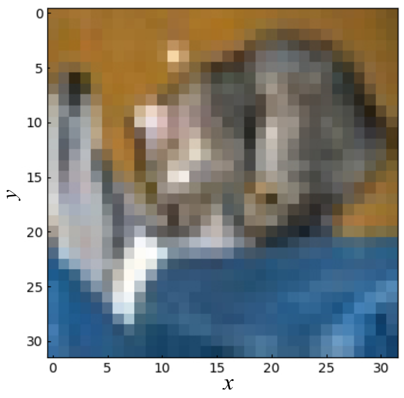

Fig. 11 Single test image from CIFAR-10 dataset. Note that the image is blurred and hard for humans to decipher. The *x* and *y* axes show the dimensions of the image (32×32).

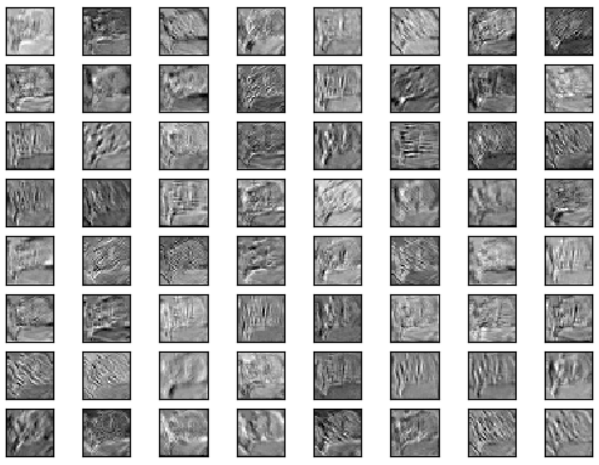

Fig. 12 Features from the second convolutional layer before RReLU with phase parameters $\alpha = \frac{1}{\sqrt{3}}$ and $\beta = -\sqrt{3}$ after the training with 20 epochs.

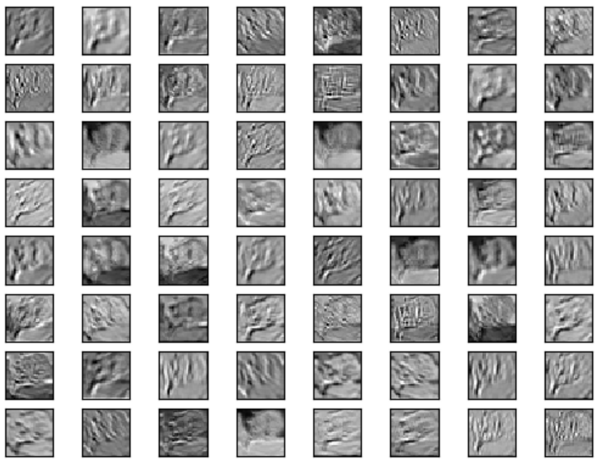

Fig. 13 Features from the second convolutional layer before ReLU after the training with 20 epochs.

show the feature maps learned right after the second convolutional layer and before the application of the activation function to the output of the convolutional layer. Figure 14 shows the feature maps from the same convolutional layer right after using the RReLU activation function with phase parameters $\alpha = \frac{1}{\sqrt{3}}$ and $\beta = -\sqrt{3}$. Note that the total number of features has quadrupled and both the negative and positive responses are being considered, leading to automatic data augmentation in the feature space. Figure 15 shows the same features after application of the ReLU activation and we can clearly see that it learns only the positive responses from the convolutional layer and hence does not lead to data augmentation in the feature space, which can explain the overfitting that is observed when using the model trained with the ReLU activation.

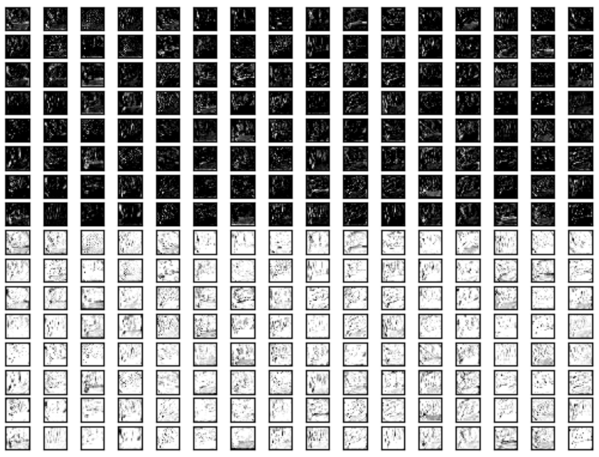

Fig. 14 Features from the second convolutional layer with RReLU and phase parameters $\alpha = \frac{1}{\sqrt{3}}$ and $\beta = -\sqrt{3}$ with 20 epochs after RReLU on convolution output.

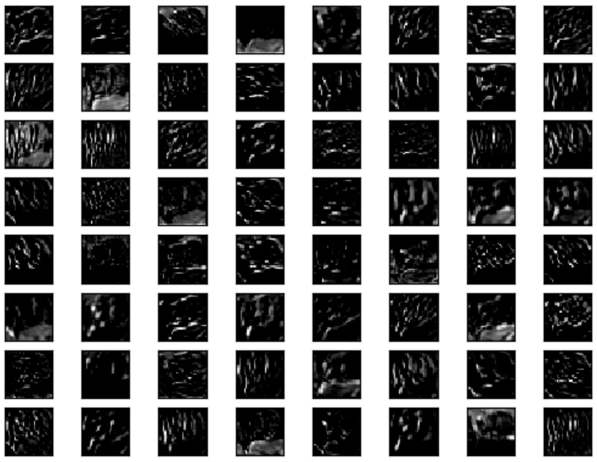

Fig. 15 Features from the second convolutional layer after ReLU with training for 20 epochs.

Till now, we have considered the RReLU activation with a fixed choice of the phase parameters. Next, we show the results of using a simple RReLU (with both phase parameters equal to 1). Figure 16 show the results of training the network with simple RReLU activation in all the layers except the dense layer. Figure 17 shows the feature map from the second convolutional layer after passing it through the simple RReLU activation after training with 20 epochs. The reader should notice the feature augmentation achieved by the reflected ReLU activation.

4.3 Result for regression

Till now, we have shown the efficacy of the reflected ReLU for the task of classification with two standard datasets. However, neural networks can also be used for the task of regression[1]. In this section, we show the efficacy of the RReLU activation for the task of

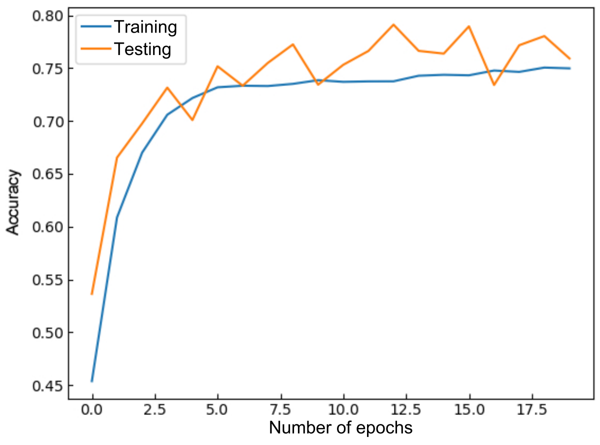

Fig. 16 Training & tesing accuracy of CIFAR-10 network with simple RReLU in all convolutional layers. Note that there is no overfitting.

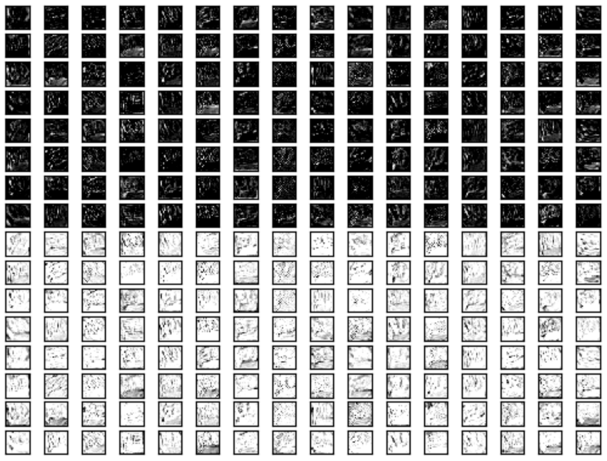

Fig. 17 Features from the second convolutional layer after simple RReLU with training for 20 epochs.

regression. In order to do that, we use a novel dataset from CFD. We choose this dataset because application of learning techniques for CFD tasks is of independent interest.

The dataset consists of readings from pressure sensors deployed on an aerodynamic projectile object as shown in Fig. 18. The flow-field around the object was governed by the non-linear in-viscid compressible Euler equations. The object executed several complex maneuvers in space as shown in Fig. 19 with a fixed free-stream pressure and velocity vector. Pressure measurements were recorded at each data point correlated with the commanded pitch and yaw of the projectile. Thus corresponding to n pressure readings at time instant t denoted by the vector P_t, the data also contains the pitch (α_t) and the yaw (β_t). Note that due to the shape of the projectile under consideration, it does not make sense to consider the roll and hence this parameter was not considered for this dataset.

The pressure was measured in the free-stream (as shown in Fig. 18) and the data contains measurements for a total of 20 000 time instances, and for each time step, there are 2000 sensor locations providing

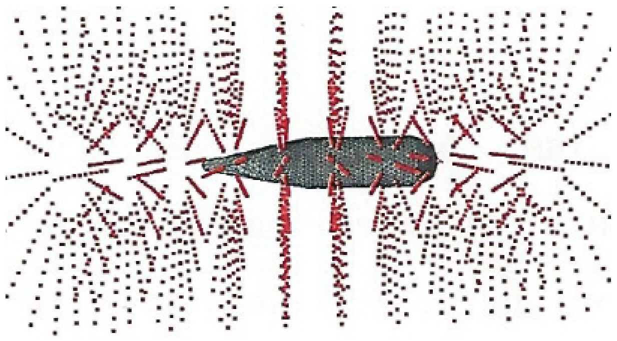

Fig. 18 Sensor locations for collecting free-stream data.

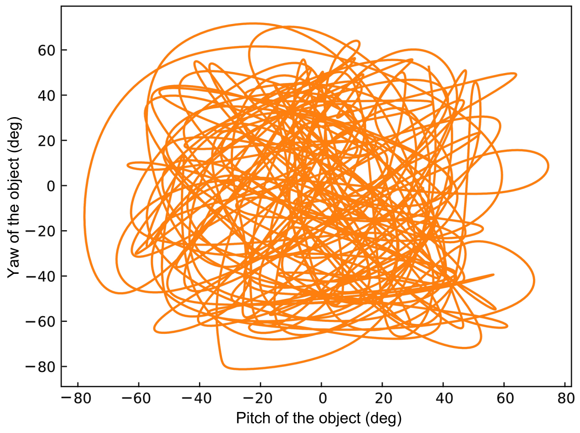

Fig. 19 Trajectory traced out by the projectile. This is obtained by plotting the pitch and yaw of the object as it moves.

pressure measurements. The input data matrix was of size 2000 $\times$ 19 800 and the measurements corresponding to the first 200 time stamps were discarded as they contained non-physical initial transients of the solution and did not provide any meaningful information.

We preprocessed the pitch and yaw angles corresponding to each time stamp t, by projecting them to the surface of an unit sphere as shown in Fig. 20. Thus for each input angle (α_t, β_t), we get the coordinates (x_t, y_t, z_t) of a point on the surface of the unit sphere. We decided to build and test a network for predicting the projected data from the pressure sensor readings. Finally, the data was standardized before being used for

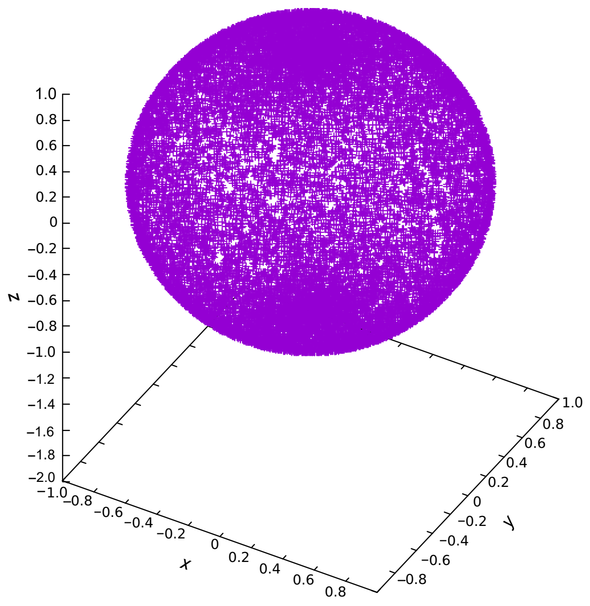

Fig. 20 Projection of pitch and yaw angles on surface of unit sphere for the free-stream data. Here *x*, *y*, and *z* axes represent the corresponding co-ordinates of the points in the cartesian system.

the training and subsequent testing.

Network architecture & results. We used a neural network architecture first reported in Ref. [48] for a feature selection algorithm with the same dataset. The network had 6 hidden layers, the number of nodes in the first and second hidden layers was the same as the number of nodes in the input layer. The architecture can be described as follows: 2000 nodes (input) are followed by 2000 nodes in the first hidden layer, which are followed by 2000 nodes in the second hidden layer, then 1000 nodes in the third hidden layer are followed by 500 nodes in the fourth hidden layer, 100 nodes in the fifth hidden layer, 10 nodes in the sixth, and finally, a layer with 3 output nodes. As reported in Ref. [48], this network achieved a training loss of 0.3334 and a validation loss of 0.3334. These numbers were reported after training for 100 epochs over several runs. The validation error was computed on 20% of the data selected at random. The network used the ReLU activation function.

For our experiments, we used the parameterized RReLU activation instead of the ReLU activation and we compared the results with those reported in Ref. [48]. As the results reported in Ref. [48] were obtained after training for 100 epochs, one of our goals was to see whether we can achieve the same (or similar) values of loss with much lesser number of epochs. The results of our experiments are shown in Table 3. We modified the network by using the parameterized RReLU activation function in all the layers except the last two layers where we used ReLU. We decided to use the ReLU activation in the last two layers as the network under consideration is deep and intuitively, we felt that using a variation of the RReLU activation in the first few layers would be enough to achieve our target. Furthermore, we ran one experiment with 10 epochs to validate our hypothesis and indeed we observed that with the parameterized RReLU in all layers except the last two layers, the performance of the network was identical in terms of the training and testing losses (identical to four places after decimal). As a result of this observation, we decided to stick with out initial network configuration where we

Table 3 Results for CFD baseline network with RRELU (phase parameters $\alpha = \frac{1}{\sqrt{3}}$ and $\beta = -\sqrt{3}$) in all layers except the last two layers after 10 epochs. Note: 20/80 test/train split. MSE: mean squared error.

Network	Train-MSE	Validation-MSE
CFD (RReLU)	0.3334	0.3329

had ReLU in the last two layers of the network. We first trained the network for only 10 epochs and used a five-fold cross validation with 20% of the data being used for the purpose of cross validation. We see that using the parameterized RReLU activation, we get the same training loss as reported by Banerjee et al.[48] within 10 epochs (in fact the network converges in the second epoch itself if we consider precision to three points after decimal). We also note that we get better validation accuracy than what was obtained in Ref. [48]. In terms of savings of the running time, this leads to a large gain in the speedup.

In order to better understand the convergence of the regressor, we finally looked at the rate of convergence of the results after 10 epochs and 50 epochs. The convergence plot for the training and testing losses after 10 epochs is shown in Fig. 21. We note that the training loss converges immediately after the first epoch and so does the testing loss. We noticed that the training and testing losses converged to four decimal points after the first epoch. Figure 22 shows the validation of the testing loss after four decimal points across the different epochs. It must be noted that convergence to four decimal places is already good enough for all practical purposes, however, we decided to show the variation in the testing loss after four decimal points in order to better understand how the testing loss behaves with the parameterized RReLU activation. Figures 23 and 24 show the training and testing losses after 50 epochs, which the testing loss being plotted to show the variation after four decimal digits. We note that the results are similar to those obtained using 10 epochs and for all practical purposes, we can conclude the training converges after 2 epochs.

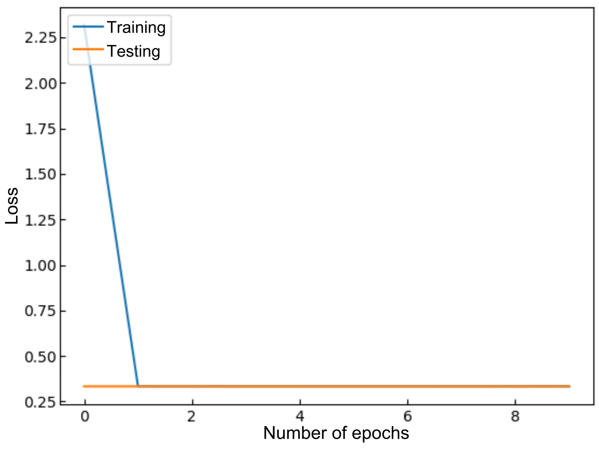

Fig. 21 Convergence of training and testing losses after 10 epochs. Note that there are variations after 2 epochs that are not visible due to scale of plot as the variations beyond four decimal places. Note: 20/80 test/train split.

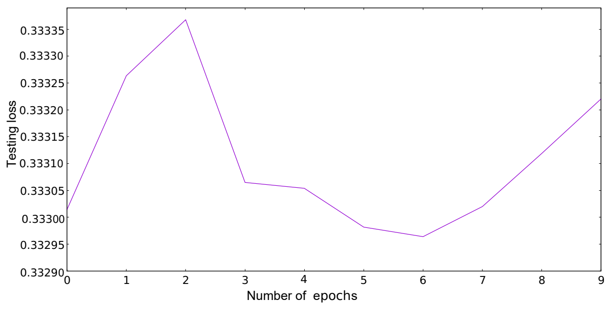

Fig. 22 Convergence of testing loss after 10 epochs. We show the variation after four decimal places in this plot. Note: 20/80 test/train split.

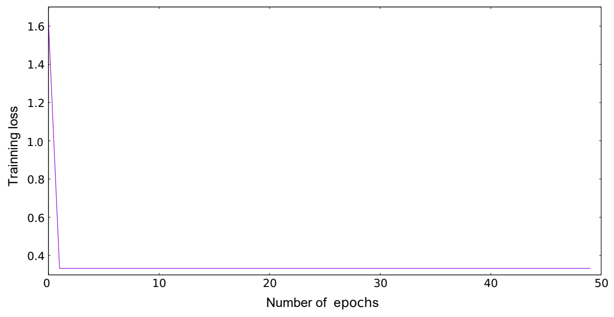

Fig. 23 Convergence of training loss after 50 epochs. Note that there are variations after 2 epochs that are not visible due to scale of plot as the variations beyond four decimal places. Note: 20/80 test/train split.

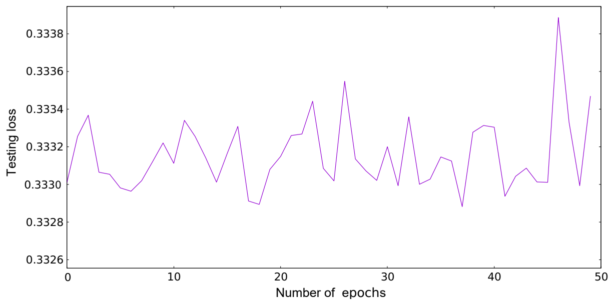

Fig. 24 Convergence of testing loss after 50 epochs. We show the variation after four decimal places in this plot. Note: 20/80 test/train split.

5 Conclusion

In this paper, we have introduced a new activation function, namely the RReLU. We have introduced and studied both a parameterized version of the activation with different "phase parameters" as well as a non-parameterized version. We did not optimize the phase parameters and simply used two phase values chosen empirically. We have shown the efficacy of the activation function for both classification as well as regression using standard and novel datasets, respectively. We have theoretically established that the RReLU activation has the reconstruction property and empirically shown that using the RReLU activation is equivalent to a form of data augmentation in the feature space where the negative responses from each layer of the network are also factored in during training and the decision making process. Finally, we have also established that the RReLU can be considered to be a form of concatenated max-min pooling. We did not study the quality of the approximations that are learned using this activation from a completely theoretical standpoint. Going forward, in another paper, we would like to study in detail the approximations that can be achieved using this class of activations.

References

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.

[2] O. M. Parkhi, A. Vedaldi, and A. Zisserman, Deep face recognition, in *Proc. British Machine Vision Conf.*, Swansea, UK, 2015.

[3] Z. Yu, T. R. Li, N. Yu, X. Gong, K. Chen, and Y. Pan, Three-stream convolutional networks for video-based person re-identification, arXiv preprint arXiv: 1712.01652, 2017.

[4] R. Socher, Y. Bengio, and C. D. Manning, Deep learning for NLP (without magic), in *Proc. Tutorial Abstracts of ACL 2012*, Jeju Island, Korea, 2012, p. 5.

[5] D. Roy, T. Mukherjee, M. Chatterjee, and E. Pasiliao, Detection of rogue RF transmitters using generative adversarial nets, in *2019 IEEE Wireless Communications and Networking Conf.*, Marrakesh, Morocco, 2019.

[6] A. Byravan and D. Fox, SE3-nets: Learning rigid body motion using deep neural networks, in *2017 IEEE Int. Conf. Robotics and Automation*, Singapore, 2017, pp. 173–180.

[7] J. Liu, Y. Pan, M. Li, Z. Y. Chen, L. Tang, C. Q. Lu, and J. X. Wang, Applications of deep learning to MRI images: A survey, *Big Data Mining and Analytics*, vol. 1, no. 1, pp. 1–18, 2018.

[8] M. Zeng, M. Li, Z. H. Fei, F. X. Wu, Y. H. Li, Y. Pan, and J. X. Wang, A deep learning framework for identifying essential proteins by integrating multiple types of biological information, *IEEE/ACM Trans. Comput. Biol. Bioinfor.*, doi: 10.1109/TCBB.2019.2897679.

[9] M. Yan, L. Liu, S. H. Chen, and Y. Pan, A deep learning method for prediction of benign epilepsy with centrotemporal spikes, in *Proc. 14th Int. Symp. Bioinformatics Research and Applications*, Beijing, China, 2018, pp. 253–258.

[10] N. Yu, Z. Yu, F. Gu, T. R. Li, X. X. Tian, and Y. Pan, Deep learning in genomic and medical image data analysis: Challenges and approaches, *J. Inf. Process. Syst.*, vol. 13, no. 2, pp. 204–214, 2017.

[11] C. Y. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals, Understanding deep learning requires rethinking generalization, arXiv preprint arXiv: 1611.03530, 2016.

[12] K. Hornik, M. Stinchcombe, and H. White, Multilayer feedforward networks are universal approximators, *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989.

[13] D. M. Loroch, F. J. Pfreundt, N. Wehn, and J. Keuper, Sparsity in deep neural networks—An empirical investigation with tensorQuant, in *Joint European Conf. Machine Learning and Knowledge Discovery in Databases*, Dublin, Ireland, 2018, pp. 5–20.

[14] W. L. Shang, K. Sohn, D. Almeida, and H. Lee, Understanding and improving convolutional neural networks via concatenated rectified linear units, in *Proc. 33rd Int. Conf. Machine Learning*, New York, NY, USA, 2016, pp. 2217–2225.

[15] M. Blot, M. Cord, and N. Thome, Max-min convolutional neural networks for image classification, in *2016 IEEE Int. Conf. Image Processing*, Phoenix, AZ, USA, 2016, pp. 3678–3682.

[16] D. Rolnick and M. Tegmark, The power of deeper networks for expressing natural functions, arXiv preprint arXiv: 1705.05502, 2017.

[17] R. M. Neal, Connectionist learning of belief networks, *Artif. Intell.*, vol. 56, no. 1, pp. 71–113, 1992.

[18] X. Glorot and Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, in *Proc. 13th Int. Conf. Artificial Intelligence and Statistics*, Sardinia, Italy, 2010, pp. 249–256.

[19] M. Courbariaux, Y. Bengio, and J. P. David, BinaryConnect: Training deep neural networks with binary weights during propagations, in *Proc. 28th Int. Conf. Neural Information Processing Systems*, Montreal, Canada, 2015, pp. 3123–3131.

[20] S. Elfwing, E. Uchibe, and K. Doya, Sigmoid-weighted linear units for neural network function approximation in reinforcement learning, *Neural Networks*, vol. 107, pp. 3–11, 2018.

[21] P. Ramachandran, B. Zoph, and Q. V. Le, Searching for activation functions, arXiv preprint arXiv: 1710.05941, 2017.

[22] Y. LeCun, Y. Bengio, and G. Hinton, Deep learning, *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[23] B. Karlik and A. Vehbi, Performance analysis of various activation functions in generalized MLP architectures of neural networks, *Int.J. Artif. Intell. Expert Syst.*, vol. 1, no. 4, pp. 111–122, 2011.

[24] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, Natural language processing (almost) from scratch, *J. Mach. Learn. Res.*, vol. 12, pp. 2493–2537, 2011.

[25] J. Turian, J. Bergstra, and Y. Bengio, Quadratic features and deep architectures for chunking, in *Proc. Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, Boulder, CO, USA, 2009, pp. 245–248.

[26] V. Nair and G. E. Hinton, Rectified linear units improve restricted Boltzmann machines, in *Proc. 27th Int. Conf. Machine Learning*, Haifa, Israel, 2010, pp. 807–814.
[27] M. D. Zeiler, M. Ranzato, R. Monga, M. Mao, K. Yang, Q. V. Le, P. Nguyen, A. Senior, V. Vanhoucke, J. Dean, et al., On rectified linear units for speech processing, in *2013 IEEE Int. Conf. Acoustics, Speech and Signal Processing*, Vancouver, Canada, 2013, pp. 3517–3521.
[28] R. Arora, A. Basu, P. Mianjy, and A. Mukherjee, Understanding deep neural networks with rectified linear units, arXiv preprint arXiv: 1611.01491, 2016.
[29] S. Goel, V. Kanade, A. Klivans, and J. Thaler, Reliably learning the ReLU in polynomial time, arXiv preprint arXiv: 1611.10258, 2016.
[30] C. Banerjee, T. Mukherjee, and E. Jr. Pasiliao, An empirical study on generalizations of the ReLU activation function, in *Proc. 2019 ACM Southeast Conf.*, Kennesaw, GA, USA, 2019, pp. 164–167.
[31] A. L. Maas, A. Y. Hannun, and A. Y. Ng, Rectifier nonlinearities improve neural network acoustic models, in *Proc. 30th Int. Conf. Machine Learning*, Atlanta, GA, USA, 2013.
[32] K. M. He, X. Y. Zhang, S. Q. Ren, and J. Sun, Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification, in *Proc. IEEE Int. Conf. Computer Vision*, Santiago, Chile, 2015, pp. 1026–1034.
[33] X. J. Jin, C. Y. Xu, J. S. Feng, Y. C. Wei, J. J. Xiong, and S. C. Yan, Deep learning with S-shaped rectified linear activation units, in *Proc. 30th AAAI Conf. Artificial Intelligence*, Phoenix, AZ, USA, 2016, pp. 1737–1743.
[34] S. Qiu and B. L. Cai, Flexible rectified linear units for improving convolutional neural networks, arXiv preprint arXiv: 1706.08098, 2017.
[35] C. Dugas, Y. Bengio, F. Bélisle, C. Nadeau, and R. Garcia, Incorporating second-order functional knowledge for better option pricing, in *Proc. 13th Int. Conf. Neural Information Processing Systems*, Denver, CO, USA, 2001, pp. 472–478.
[36] L. Trottier, P. Giguere, and B. Chaib-draa, Parametric exponential linear unit for deep convolutional neural networks, in *Proc. 16th IEEE Int. Conf. Machine Learning and Applications*, Cancun, Mexico, 2017, pp. 207–214.
[37] B. Grelsson and M. Felsberg, Improved learning in convolutional neural networks with shifted exponential linear units (ShELUs), in *Proc. 24th Int. Conf. Pattern Recognition*, Beijing, China, 2018, pp. 517–522.
[38] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio, Maxout networks, arXiv preprint arXiv: 1302.4389, 2013.
[39] H. W. Lin, M. Tegmark, and D. Rolnick, Why does deep and cheap learning work so well? *J. Stat. Phys.*, vol. 168, no. 6, pp. 1223–1247, 2017.
[40] P. Petersen and F. Voigtlaender, Optimal approximation of piecewise smooth functions using deep ReLU neural networks, *Neural Networks*, vol. 108, pp. 296–330, 2018.
[41] Z. Yu, T. R. Li, N. Yu, Y. Pan, H. M. Chen, and B. Liu, Reconstruction of hidden representation for robust feature extraction, *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 2, p. 18, 2019.
[42] Z. Yu, N. Yu, Y. Pan, and T. R. Li, A novel deep learning network architecture with cross-layer neurons, in *2016 IEEE Int. Conf. Big Data and Cloud Computing, Social Computing and Networking, Sustainable Computing and Communications*, Atlanta, GA, USA, 2016, pp. 111–117.
[43] P. Ballester and R. M. Araujo, On the performance of GoogLeNet and AlexNet applied to sketches, in *Proc. 30th AAAI Conf. Artificial Intelligence*, Phoenix, AZ, USA, 2016, pp. 1124–1128.
[44] A. Kendall, M. Grimes, and R. Cipolla, PoseNet: A convolutional network for real-time 6-DOF camera relocalization, in *Proc. IEEE Int. Conf. Computer Vision*, Santiago, Chile, 2015, pp. 2938–2946.
[45] A. Mahendran and A. Vedaldi, Understanding deep image representations by inverting them, in *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, Boston, MA, USA, 2015, pp. 5188–5196.
[46] D. P. Kingma and J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980, 2014.
[47] A. Krizhevsky and G. Hinton, Convolutional deep belief networks on CIFAR-10, *Unpublished Manuscript*, vol. 40, no. 7, pp. 1–9, 2010.
[48] C. Banerjee, T. Mukherjee, C. Lilian, D. Reasor, X. W. Liu, and E. Pasiliao, A feature selection algorithm using neural networks, *International Journal of Machine Learning and Computing*, vol. 4, pp. 1–8, 2020.

Chaity Banerjee received the MS degree and PhD degree in computer science from Florida State University both in 2017. Currently she is a postdoctoral associate in the Department of Industrial Engineering at the University of Central Florida. Her research interests include machine learning, data analysis, and feature segmentation and localization in big data including Cryo-EM tomography and single particle electron microscopy.

Tathagata Mukherjee received the MS degree and PhD degree in computer science from Florida State University in 2014 and 2016, respectively. Currently he is an assistant professor in computer science at the University of Alabama in Huntsville. He has worked extensively with software defined radios with applications to assured communication and passive sensing. His research interests are broadly in the areas of cyber-security and network forensics where he is interested in applying machine learning, data analytics, and optimization techniques for offensive security, malware analysis, and internet forensics. He is also interested in quantum computing and its applications, the theory and application of deep learning, and graph theory.

Eduardo Pasiliao Jr. received the BS degree in mechanical engineering from Columbia University, New York, NY, USA, ME degree in coastal and oceanographic engineering, and PhD degree in industrial and systems engineering from the University of Florida, Gainesville, FL, USA in 1992, 1995, and 2003, respectively. He is a senior research engineer at the Air Force Research Laboratory Munitions Directorate (Eglin AFB FL) and is the director of the AFRL Mathematical Modeling and Optimization Institute. His research interest is in mathematical optimization with emphasis on social and communication networks.