Soft++, a multi-parametric non-saturating non-linearity that improves convergence in deep neural architectures

Andrei Ciuparu\textsuperscript{a,b}, Adriana Nagy-Dăbăcan\textsuperscript{a}, Raul C. Mureşan\textsuperscript{a,*}

\textsuperscript{a} Department of Experimental and Theoretical Neuroscience, Transylvanian Institute of Neuroscience, Str. Pioi 33, Cluj-Napoca 400157, Romania
\textsuperscript{b} Basis of Electronics Department, Technical University of Cluj-Napoca, Str. G. Bariuș 26-28, Cluj-Napoca 400027, Romania

A R T I C L E   I N F O
Article history:
Received 24 October 2018
Revised 18 October 2019
Accepted 6 December 2019
Available online xxxx

Communicated by Prof. M. Bianchini

Keywords:
Activation function
Deep learning
Error backpropagation
Multi-layer perceptron
Convolutional neural network

A B S T R A C T
A key strategy to enable training of deep neural networks is to use non-saturating activation functions to reduce the vanishing gradient problem. Popular choices that saturate only in the negative domain are the rectified linear unit (ReLU), its smooth, non-linear variant, Softplus, and the exponential linear units (ELU and SELU). Other functions are non-saturating across the entire real domain, like the linear parametric ReLU (PReLU). Here we introduce a nonlinear activation function called Soft++ that extends PReLU and Softplus, parametrizing the slope in the negative domain and the exponent. We test identical network architectures with ReLU, PReLU, Softplus, ELU, SELU, and Soft++ on several machine learning problems and find that: i) convergence of networks with any activation function depends critically on the particular dataset and network architecture, emphasizing the need for parametrization, which allows to adapt the activation function to the particular problem; ii) non-linearity around the origin improves learning and generalization; iii) in many cases, non-saturation across the entire real domain further improves performance. On very difficult learning problems with deep fully-connected and convolutional networks, Soft++ outperforms all other activation functions, accelerating learning and improving generalization. Its main advantage lies in its dual parametrization, offering flexible control of the shape and gradient of the function.

© 2019 The Author(s). Published by Elsevier B.V.

This is an open access article under the CC BY license. (http://creativecommons.org/licenses/by/4.0/)

1. Introduction
Deep learning has become extremely popular in recent years [30,43]. Deep neural networks, trained on vast amounts of data, have reached performance levels that apparently match or surpass human expertise on difficult tasks, such as object recognition [25] or complex games, like Go [45]. Such astonishing progress has been enabled by three key factors: the availability of very large amounts of training data [42], the vigorous increase in computing power due to massively parallel cores of modern GPUs [43], and by transiting from shallow to deeper neural architectures [28,2]. The present study will focus on this latter breakthrough.

The enthusiastic development of artificial neural networks in the 70’s and 80’s has hit a hard wall when it was recognized that it is very difficult to train networks with more than one or two hidden layers [2]. This limitation arises due to the so-called vanishing gradient problem in the error backpropagation algorithm [17,3], whereby the gradient decreases exponentially fast as it is backpropagated through a multi-layered neural network. The problem is exacerbated when classic activation functions are used, like the sigmoid or hyperbolic tangent (tanh), whose gradients vanish asymptotically as the input departs from origin [38].

More recently, the vanishing gradient problem has been overcome by using two strategies. First, adaptive training algorithms emerged that do not use the value of the gradient but only its sign to update weights in the network – with the most successful algorithms being the RPROP [39] and its improved variant, IRPROP [19,20]. A second improvement was to use non-saturating activation functions like the Rectified Linear Unit (ReLU) [14,34]. The latter is the most popular activation function in deep learning architectures [30,6] because its gradient remains constant throughout the positive domain. These two strategies enable training of networks with unprecedented depth.

It is important to be able to train deep networks because their representational power surpasses that of shallow ones [2]. Indeed, by studying the complexity of the function implemented by networks of various depths, it has been shown that deeper networks are able to implement more complex functions than shallow ones [4]. It was further demonstrated that shallow networks are incapable of providing a good approximation of almost any function
unless a sufficient number of neurons is used [26]. Thus, robust training of deep networks is desirable and, as we will show, the properties of the activation function are very important for the training process.

The popular activation functions, like the ReLU and its continuous counterpart, Softplus [9], are non-saturating in the positive domain, but their value and gradient is, or approaches zero in the negative domain – i.e. they are only partially non-saturating (Fig. 1). The recovery of saturated units, although possible, is very slow, creating large plateaus during learning [10]. In addition, reduced gradients further contribute to the vanishing gradient problem, especially when classical backpropagation is used. A solution to this issue has been introduced with the leaky ReLU (LReLU), which adds a mild linear slope in the negative domain, and its parametrized version, the parametric ReLU (PReLU) [15], both yielding fully non-saturating activation functions. Another recent activation function, the exponential-linear unit (ELU) is saturating in the negative domain but it takes both positive and negative values, constraining average activations around zero and reducing the bias shift effect [8]. ELU networks can greatly accelerate learning [8]. An additional variant of ELU, the scaled exponential-linear unit (SELU) further provides automatic normalization within the network by preserving zero mean and unit variance [23].

Here, we show that convergence speed in deep neural architectures with various activation functions is strongly problem-dependent. To this end, we use extensive testing on both large and small, difficult datasets, with both fully-connected (MLP) and convolutional networks, combined with gradient-based or adaptive learning strategies. We argue that further improvement is possible over existing activation functions by introducing a bi-parametric, fully customizable activation function called Soft++. The latter is based on the Softplus function and allows one to parametrize the slope in both the negative and positive domains. Soft++ shares the advantages of ELU but it can be adapted better to a particular learning problem, thus outperforming all activation functions in terms of convergence speed and, often, test accuracy.

2. Methods

2.1. Activation functions

We compared the performance of deep networks having six activation functions: ReLU, Softplus, PReLU (LReLU), ELU, SELU, and Soft++. The ReLU function is the most popular activation function [30]. It is linear in the positive domain and zero in the origin and over the negative domain:

\[
ReLU(x) = \begin{cases} 
0, & x \leq 0 \\
 x, & x > 0 
\end{cases}
\] (1)

Fig. 1. Activation functions (a) and their derivatives (b), \(c = 10, k = 1, \alpha = 0.1\). Parameters of Soft++ can be chosen to obtain a unitary gradient in the origin - (c) and (d). Unlike ReLU, PReLU, ELU, and SELU, Soft++ has smooth gradients across the entire real domain.
A “soft” variant of ReLU is Softplus [9]:

$$\text{Softplus}(x) = \ln(1 + e^x)$$

(2)

Both ReLU and Softplus are saturating in the negative domain (Fig. 1). A non-saturating version of ReLU was recently introduced [15]:

$$\text{PReLU}(x) = \begin{cases} a \cdot x, & x \leq 0 \\ x, & x > 0 \end{cases}$$

(3)

where, $a$ is a coefficient determining the slope of the negative part. In PReLU, the parameter $a$ is learned together with the weights in the network. If the parameter is fixed, PReLU becomes equivalent to LeRelu. Here, we fixed the parameter in each experiment.

ELU falls in the category of saturating activation functions, but takes both positive and negative values and it is locally non-saturated around the origin [8]:

$$\text{ELU}(x) = \begin{cases} \alpha \cdot (e^x - 1), & x \leq 0 \\ x, & x > 0 \end{cases}$$

(4)

where, $\alpha$ is a parameter determining the saturating value in the negative domain, usually set to 1 [8]. The gradient of ELU in the origin is $\alpha$. Here, we fixed the value of $\alpha$ to 1 in all experiments. SELU is the scaled variant of ELU, as it introduces a second, scaling parameter, $\lambda$ [23]:

$$\text{SELU}(x) = \lambda \cdot \begin{cases} \alpha \cdot (e^x - 1), & x \leq 0 \\ x, & x > 0 \end{cases}$$

(5)

where, $\alpha$ is set to $-1.67326$ and $\lambda$ is set to $-1.05070$ for standard scaled inputs.

Here, we propose a new, fully non-saturating function called Soft++, extending both Softplus and PReLU, and defined as follows:

$$\text{Soft++}(x) = \ln(1 + e^{kx}) + \frac{x - \ln(2)}{c}$$

(6)

where, $k$ is an exponent coefficient and $c$ is a slope coefficient which sets the slope in the negative domain. The constant $\ln(2)$ term removes the offset of the Soft++ function in the origin. The derivative of Soft++ is:

$$\text{Soft++}'(x) = \frac{(c \cdot k + 1) \cdot e^{kx} + 1}{c \cdot (e^{kx} + 1)}$$

(7)

For positive $k$ and $c$, the gradient of Soft++ in the negative domain is inversely proportional to the slope coefficient, $c$, i.e. it converges to $1/c$. In the positive domain, the gradient of Soft++ converges to $(ck+1)/c$ (Fig. 1). When $k$ and/or $c$ is negative, exotic shapes of the activation function can be obtained.

By choosing the parameters $k$ and $c$ appropriately, the gradient of Soft++ in the origin can be matched to that of ELU (Fig. 1c and d). For later tests we found that matching the initial gradients (given that weights are narrowly distributed around 0) creates better conditions to compare activation functions because learning starts under similar conditions (see below for CIFAR-100). Unlike most other activation functions (except Softplus), Soft++ is differentiable and has a smooth gradient across the entire real domain (see Fig. 1b and d).

2.2. Network architecture

To test the performance of the various activation functions, we used: i) fully-connected multi-layered neural networks (MLP) with 6–10 hidden layers, a similar depth as used by Cireșan [7]; ii) deep convolutional neural networks with 7–11 layers (CNN). In all cases we maintained the architecture of the network fixed and created different copies with the different activation functions.

For fully-connected MLP networks, hidden layers contracted from the input towards the output, set as a percentage of the size of the input layer. These were set as follows, from input to output: for MNIST dataset (6 layers: 60%, 50%, ... , 20%, 10%), for the visual cortex dataset (10 layers: 90%, 70%, 50%, 30%, 20%, ..., 20%), for the synthetic dataset (10 layers: 110%, 100%, 90%, ..., 30%, 20%). In all cases, an additional Softmax layer was added at the output in order to minimize cross-entropy [11]. The networks thus acted as probabilistic classifiers: Each output was constrained between 0 and 1, the value of output $i$ representing the probability that a sample from class $i$ was presented as input. A sample was labeled as belonging to the class whose corresponding output was largest (argmax).

Initial bias weights were drawn from a Gaussian distribution as follows: 0.5 ± 0.1 for the MNIST and synthetic dataset, and 0.0 ± 0.1 for the visual cortex dataset. In MLP networks, neuron-neuron weights were initialized as (modified LeCun uniform):

$$w_{ij} = s \cdot U(-1.0, 1.0) \cdot \sqrt{\frac{3}{\text{Fan-in}(i)}}$$

(8)

where, $w_{ij}$ is the weight from neuron $j$ to neuron $i$, $s$ is a scaling factor, $U$ denotes the uniform distribution, and $\text{Fan-in}(i)$ is the number of inputs neuron $i$ receives from its afferents. The scaling parameter, $s$, was set to 0.1 for the MNIST and visual cortex datasets, and 0.5 for the synthetic dataset. For deep CNNs, initialization of weights was performed as described in Glorot and Bengio [10]. In addition, when the SELU function was used in MLP networks, weights were initialized using the LeCun normal initializer [32]. In CNN networks with SELU activation function we implemented both the initializer used for the other functions (to have an exact matching of the testing conditions) and the LeCun normal initializer.

For tests on deep CNNs, we used two different network architectures, tested on CIFAR-10 and CIFAR-100 datasets. First, we implemented a version of the well-known network AlexNet [25], modified in order to work on the CIFAR-10 dataset (https://github.com/Natsuf7677/Modified-AlexNet-Tensorflow). This network is a deep convolutional neural network that consists of four convolutional layers (C [kernel size x number of kernels, stride] and same padding) with local response normalization (Lrn with radius = 5; alpha = $10^{-4}$; beta = 0.75, bias = 2) and pooling (P with height = 3; width = 3; stride = 2 and valid padding), followed by three fully connected layers (Fc [number of nodes, dropout]).

The last one having a Softmax activation. The network we used has the following: C1 [11 × 96,2] Lrn1, P1, C2 [5 × 256,1 Lrn2, P2, C3 [3 × 384,1], C4 [3 × 256,1], P3, Fc1 [1024,0.5, Fc2 [2048,0.5], Fc3[10,0.] (Softmax). Weights of all layers were initialized as random normal variables with mean 0 and standard deviation of 0.01.

The biases were initialized to 0 for C1 and C3 and 1 for all other layers.

Second, for tests on both CIFAR-10 and CIFAR-100 we developed a stacked network similar to that in [8], but simpler, consisting of stacks of two convolutional layers followed by a pooling and dropout layer. The convolutional layers in each stack had the same number of kernels (kernel height x kernel width x number of kernels): [3 x 3 x 256, 3 x 3 x 256, [3 x 3 x 256, 3 x 3 x 256, [2 x 2 x 512, 2 x 2 x 512, [2 x 2 x 512, 2 x 2 x 512, [2 x 2 x 1024, 1 x 1 x 1024] followed by one fully connected output layer with Softmax activation. The dropout applied after the pooling was different for each stack and was greater as the kernel count increased: 0.1, 0.15, 0.2, 0.25, and 0.3. All weights were initialized using the uniform initialization method described in Glorot and Bengio [10], and all biases were set to 0.001. For the SELU activation function we also used a version of the network where weights were initialized using the LeCun normal technique [32], coupled with $\alpha$-dropout [23], applied only to the last two stacks.
when α-dropout was applied to the same stacks like for Soft++, SELU networks did not converge.

2.3. Benchmark datasets

2.3.1. MNIST datasets

For benchmarking, we used multiple datasets. First, we used the well-known MNIST database [31] with 70,000 samples of 10 hand-written digits, which is a standard benchmark for many machine learning techniques. The database contains 60,000 training and 10,000 test images, the sets being disjoint [29]. During training, only samples in the training set were presented to the networks, in each training step. The test set was split evenly into 5,000 validation and 5,000 test images.

2.3.2. CIFAR-10 and CIFAR-100

For convolutional networks, we used other well-known datasets, CIFAR-10 and CIFAR-100 [8,24]. These two databases contain a total of 60,000 32 × 32 × 3 images split evenly into 10 and 100 classes, respectively. Datasets are pre-divided into a training set of 50,000 images and a test set of 10,000 images, each having all classes equally represented. As with the MNIST dataset, only the training images were used for training networks while the test set was split evenly into 5,000 validation and 5,000 test images.

2.3.3. VISUAL cortex data

In many fields of science, where machine learning is used to discover structure in the data, having many examples per class is simply impossible, due to practical limitations and, often, the number of features can exceed the number of examples. Here, we experimented on such a dataset generated from in vivo recordings performed in visual cortex of mice. Multi-electrode extracellular electrophysiology signals were collected using Neuronexus silicon probes (A-type, 4 × 2 tetrode configuration) inserted into the visual cortex of isolancan anesthetized mice (1.2% mixed with O2). Animal manipulation and recordings were done in accordance with Romanian law, European Communities Council Directive 2010/63/EU regarding the care and use of animals for experimental procedures, and Society for Neuroscience guidelines and approved by the Local Ethics Committee (3/CE/02.11.2018) and the National Veterinary Authority (ANSVSA: 147/04.12.2018). The 32 signals were sampled at 32 kHz/s, amplified 10,000 × and digitized. Visual stimuli consisted of full-field drifting gratings (0.11 cycles/deg; 1.75 cycles/s; contrast: 25%, 50%, and 100%; 8 directions in steps of 45°; 10 trials per contrast-direction combination, 240 trials in total; stimulus presentation duration: 2600 ms). These visual stimuli drive cortical neurons vigorously [18]. Deep CNNs trained on images usually develop similar, grating-like receptive fields [25]. Data was filtered offline (band-pass 500 Hz–3500 Hz) and spike-sorting [33] was performed to isolate the firing of individual neurons (single units).

After preprocessing and spike sorting the data, we obtained the spike (firing) times of 30 isolated visual cortex neurons. For each trial, we segmented the 2600 ms long recording into twenty 130-ms long windows and counted how many spikes each neuron fired within each window. A vector of 20 × 30 = 600 features was then compiled by concatenating all windows for all neurons. A number of 4 output classes were assigned to the trials by considering the 4 orientations (0–180°, 45–225°, 90–270°, 135–315°) of the visual drifting grating [21] presented to the animal. Thus, the final dataset used for experimenting with networks consisted of 240 samples with 600 features each and distributed evenly across 4 classes (60 samples per class). The dataset was then randomly split into 120 training examples (50%), 60 validation (25%) and 60 (25%) test samples. The split always balanced across the 4 classes. A final normalization was performed on the dataset by mapping all values to the interval [−0.1, 0.1] (min-max normalization). The visual cortex dataset is freely available here: http://raulmuresan.ro/sources/softplusplus/VisualCortexData.zip

2.3.4. Synthetic dataset

For a final batch of tests we created a synthetic dataset with a very low signal-to-noise ratio, where a majority of features was irrelevant. The rationale behind this choice was that most machine learning techniques suffer from the presence of irrelevant features [37,27,1,13], while deep learning was proposed to alleviate the problem [30], at least in some cases [5,40]. In addition, this dataset resembles well the typical case found in many fields of science: large feature set, few classes and few examples. We chose an extreme case, whereby the dataset contained 100-dimensional samples but only 1 dimension contained information about the actual class. The 99 non-informative dimensions were random numbers drawn from a uniform distribution:

\[ ND = U(−0.5, 0.5) \]

The informative dimension was a function of the class and some added white noise, as follows:

\[ ID_i = \frac{i}{10} - 0.5 + U(−0.02, 0.02) \]

where, \( i \) is the class to which the sample belongs.

We chose 10 classes and for each class generated 100 samples, yielding a dataset with 1000 samples in total. This synthetic benchmark dataset is available for download here: http://raulmuresan.ro/sources/softplusplus/ReferenceData.zip. For each training/testing, the synthetic dataset was randomly split in three disjoint subsets: training (60%), validation (20%), and testing (20%).

2.4. Tests setup

For each test, we created multiple identical networks as described in Section 2.2, such that only the activation function differed across networks. Only for the case of SELU, we also created identical architectures but used a different weight initializer and dropout technique to favor self-normalization [23]. The parameters \( a \), for PReLU, and \( k \) and \( c \), for Soft++, were not adjusted during learning but kept fixed for each instantiation.

2.4.1. MLP networks

MLPs were trained for 300, 500, and 1000 steps, for the MNIST, visual cortex, and synthetic dataset, respectively, using the batch IRPROP+ algorithm [19]. For the visual cortex and synthetic dataset we applied full batch training (entire training set learned in each epoch). For MNIST, to speed up training, only 10% of the training set was randomly selected for each training step (like in mini-batch) but every 10 training steps the full training set was presented to the network. This strategy allowed for a fast monitoring of convergence and also yielded good training results. To increase the robustness of training and the generalization ability, we used 5% dropout probability [16] for the synthetic dataset and 40% for the visual cortex dataset. No dropout was used for MNIST. Because performance can vary as a function of the random splitting of the dataset and of the initial weights of the network, we created between 20 and 50 networks for each test. To match the exact initial conditions across networks, instantiations were in some cases performed by fixing the random seed for obtaining the same network but with different activation functions – this however did not seem to yield different results from those using individual randomizations. The visual cortex and synthetic datasets were also randomly split into training/validation/test sets for each network. For the MNIST dataset random splitting was not applied because this was already split into disjoint training and test sets [29] – i.e., we
chose a fixed splitting into 60,000/5,000/5,000 samples for training/validation/test.

2.4.2. Convolutional neural networks

For the Alexnet on CIFAR-10 we used for training the Adam optimization [22] with a mini-batch size of 100 samples. We initially screened the networks for 200 epochs on the validation set and determined that performance already saturated by the 100th epoch. As such, all tests presented here were run for 100 epochs. In the present study, we were not interested in maximizing performance on CIFAR-10 but rather to compare how identical networks with different activation functions fared on this hard problem. Networks saturated at classification performances close to those reported in [24], except networks with Soft++, which significantly surpassed them.

In an additional set of tests on CIFAR-10, we used the more expressive deep CNN (stacked network modified after [8]) to further boost performance to values exceeding 90%. In that case, we only tested networks endowed with ELU, SELU and Soft++ activation functions. For SEU, we used the required adaptations (LeCun initialization and α dropout applied to the last two stacks) [23].

For the deep CNN networks on CIFAR-100 we wanted to replicate the previous study on ELU networks [8], as well as to test the different activation functions in conditions as similar as possible in order for the comparison to be fair. As such, we preprocessed the data using global contrast normalization [12], and instead of the ZCA, we used a difference of Gaussians filter in order to extract contrast information from the image [35]. We also imposed a constant learning rate of 0.005 and used momentum optimization with a parameter of 0.9. Images were rescaled to the interval −0.1 to 0.1 for all networks (min-max). All functions were tested under identical training conditions, with a minibatch size of 100 and were run for 500 or 1500 epochs of training. We measured performance on the training and validation sets every epoch, and after the training was complete, we measured performance on the test set with the frozen network. Due to the high computational demand, we only ran three trials for each activation function. For SELU networks, we ran two different simulations: (i) preserving all parameters identical to those used with the other activation functions (SELU unadapted); (ii) adapting the initialization and the dropout technique to favor self-normalization, as described in [23].

For different datasets and network architectures, we performed a hyperparameter search for determining the parameters of Soft++ yielding the best performance. This search process was always done by considering the performance on the validation set only.

Network parameters, training algorithms and information about datasets are given in Table 1.

3. Results

3.1. Performance and learning curves on MNIST dataset

We first evaluated how networks with identical architecture but different activation functions fared on the standard MNIST dataset, and focused on the convergence speed. Initially, we tested ReLU, PReLU, Softplus, ELU, and three Soft++ networks (c = 10, 30, and 50, with k = 1) using the custom weight initialization (see Eq. (8)) and raw inputs. Among these networks, Soft++ and ELU performed the best, clearly surpassing ReLU and Softplus, and exceeding 98% accuracy on the test set (Fig. 2a). We also quantified the number of training steps required to reach a performance of 50% on the training and validation sets (Fig. 2b, left and right, respectively). In this first network group, Soft++ reached a performance of 50% in the fewest steps (Fig. 2b–d).

Next, we tested the performance of SELU networks on MNIST. We kept the same architecture but changed the initialization (LeCun normal) and standardized the inputs, to favor self-normalization [23]. Indeed, SELU networks surpassed the previous network group on the test set (Fig. 2a, right), but tended to overfit more, reaching close to 100% on the training set (Fig. 2a, left). To have a fair comparison to Soft++, we also tested these latter networks with c = 100 and k = 1.2 using the same weight initialization and standardized input like we used for SELU. We found that this new Soft++ network easily reached a performance similar to SELU, being slightly better on the validation set (Fig. 2a, middle), a bit below on the test set (Fig. 2a, right), and overfitting less than SELU (Fig. 2a, left).

All networks, except the saturating ReLU and Softplus, exceeded 98% accuracy on the test set (Fig. 2a). Results on MNIST indicate that Soft++ can be flexibly adapted to different weight initialization schemes and different input normalizations to boost performance.

3.2. Performance and learning curves on CIFAR-10 dataset

We next tested the deep Alexnet CNN trained on the CIFAR-10 dataset (Fig. 3a–e). Similarly as before, the saturating ReLU and Softplus networks performed worst—which they reached a maximum performance of 63.95±1.75% and 63.93±1.33%, respectively (Fig. 3a and 3e). Indeed, their performance increased significantly slower than for other networks (Fig. 3d; p < 0.01; heteroscedastic, double tailed t-test, Bonferroni corrected). PReLU networks (Fig. 3b) reached a maximum performance of 65.89±0.25%, converged significantly faster than both but slower than Soft++ (see Fig. 3d). For both PReLU and Soft++ we tested convergence as a function of their parameter (a and c, respectively) and found that the steeper these functions were in the negative domain (larger a and smaller c), the faster was the convergence (Fig. 3b and c). For Soft++, we always fixed k to 1. ELU networks converged faster than ReLU and Softplus but similarly to PReLU (Fig. 3a and d), however they surpassed the former three on the test set (67.17±0.81%) after training (Fig. 3e). Finally, Soft++ networks with c = 0.5 converged the fastest of all network types and reached a maximum performance of 68.28±0.1%, surpassing all other networks (p < 0.05).

When trained for only 100 epochs, the AlexNet reached a rather modest performance on the CIFAR-10 dataset. However, for the sake of diversity, we wanted to estimate learning dynamics with various activation functions on a well-known dataset (CIFAR-10) and with a well-known network architecture (AlexNet). Indeed, AlexNet is not optimized to learn on CIFAR, therefore we next resorted to the deep stacked CNN, inspired by a network that was optimized for ELU [8]. This network performed much better than Alexnet on CIFAR-10 (Fig. 3f and g): ELU and Soft++ (c = 1; k = 2) reached a performance on test set > 91.3%, but there was no significant difference between the two. With this network we also tested SELU (using LeCun normal weight initialization and α-dropout) and found that it performed worse than the other two activation functions on test set (89.12%). Indeed, while the dynamics of learning on the train set was similar across the three functions (Fig. 3f), on the validation set SELU networks performed significantly worse (Fig. 3g). Next, we used the same network architecture on a much harder problem: the CIFAR-100 dataset.

3.3. Performance and learning curves on CIFAR-100 dataset

CIFAR-100 contains 10 × more classes than CIFAR-10 but a similar number of examples. Therefore, learning on this dataset was expected to be much more difficult. On CIFAR-100 ELU networks were shown to reach a performance of 70–75% using a very deep CNN network, global contrast normalization and ZCA, dynamic learning rate schedule and adjusted dropout [8]. To be able to isolate the effect of the learning function, we resorted to a lighter,
Table 1
Parameters of networks, details about training algorithms, and information about datasets.

| Parameter name | Value |
|----------------|-------|
| **MLP architecture** | | |
| Activation function | ReLU, PReLU, Softplus, ELU, SELU, Soft++ |
| Soft++ slope coeff. \(c\) | 2, 10, 30, 50 (depends on experiment) |
| Soft++ exp. coeff. \(k\) | 1, 1.8, 1.93 (depends on experiment) |
| PReLU slope coeff. \(a\) | 0.02 (unless specified) |
| Input layer size | 784 (MNIST), 600 (Vis. Cortex & Synthetic) |
| Number of hidden layers | 6 (MNIST), 10 (Vis. Cortex & Synthetic) |
| Hidden layer sizes (% of input layer size) | MNIST: 60, 50, 40, 30, 20, 10 |
| **CNN architecture** | | |
| **Alexnet:** | C1 [11 × 96,2] |
| **Conv layer:** (C [kernel size x number of kernels, stride] and same padding) | Lrn1, P1, C2 [5 × 256,1], Lrn2, P2, C3 [3 × 384,1], C4 [3 × 256,1], P3, Fc1 [1024, 0.5], Fc2 [2048,0,5], Fc3[10,0] (softmax) |
| **Local Response Normalization:** (Lrn radius = 5; alpha = 10⁻²; beta = 0.75; bias = 2) | See: https://github.com/Natsu6767/Modified-AlexNet-Tensorflow |
| **Pooling:** (P: height = 3; width = 3; stride = 2; valid padding), | C1 [3 × 256,1], C2 [3 × 256,1], P1, D1 [0.1], C3 [3 × 256,1], Lrn1, C4 [3 × 256,1], Lrn2, P2, D2 [0.15], C5 [2 × 512,1], Lrn3, C6 [2 × 512,1], Lrn4, P3, D3 [0.2], C7 [2 × 512,1], C8 [2 × 512,1], P4, D4 [0.25], C9 [2 × 1024,1], C10 [1 × 1024,1], P5, D5 [0.3], Fc1[100,0] (softmax) |
| **Fully Connected layer:** (Fc [number of nodes, dropout]) | | |
| **Stacked net:** | | |
| **Conv layer:** (C [kernel size x number of kernels, stride = 1] and same padding) | | |
| **Local Response Normalization:** (Lrn: radius = 3, alpha = 1, beta = 5, bias = 1) | | |
| **Pooling:** (P: height = 2; width = 2; stride = 2; same padding), | | |
| **Dropout:** (D: [dropout probability]) | | |
| Fully Connected layers (Fc [number of nodes, dropout]); \(\alpha\)-dropout was applied to the last two stacks for the adapted SELU test | | |
| **MNIST dataset** | | |
| Sample dimensionality | 784 (28 × 28 grayscale pixels) |
| Number of samples | 70,000 |
| Number of classes | 10 |
| Training set size | 60,000; 6,000 mini batch + full-batch every 10 steps |
| Validation set size | 5,000 |
| Test set size | 5,000 |
| **CIFAR-10/100 dataset** | | |
| Sample dimensionality | 3,072 (32 × 32 color pixels) |
| Number of samples | 60,000 |
| Number of classes | 10/100; 100 mini-batch size |
| Training set size | 50,000 |
| Validation set size | 5,000 |
| Test set size | 5,000 |
| **Visual cortex dataset** | | |
| Sample dimensionality | 600 |
| Number of samples | 240 |
| Number of classes | 4 |
| Training set size | 50%, full batch training |
| Validation set size | 25% |
| Test set size | 25% |
| **Synthetic dataset** | | |
| Sample dimensionality | 100 |
| Number of samples | 1,000 |
| Number of classes | 10 |
| Training set size | 600; 100 mini batch + full-batch every 10 steps |
| Validation set size | 200 |
| Test set size | 200 |
| **Training parameters for MLP, Vis. Cortex & Synthetic** | | |
| Training algorithm | Batch & Minibatch, IRPROP⁺, Dropout probability | |
| \(\Delta_0\) | 0.001 / Fan-in |
| \(\Delta_{\text{min}}\) | MNIST: 1e-15, Vis. Cortex & Synthetic: 1e-9 |
| \(\Delta_{\text{max}}\) | MNIST: 0.01, Vis. Cortex & Synthetic: 0.1 |
| \(\beta\) | MNIST: 0.95, Vis. Cortex & Synthetic: 0.9 |
| \(\beta\) | 1.05 |
| MNIST: 0%, Vis. Cortex: 40%, Synthetic: 5% |
| **Training parameters for CNN** | | |
| Training algorithm | Minibatch; Adam (Alexnet); Momentum 0.9 (CIFAR-100) |
| Batch size | 100 |
| Learning rate | Alexnet: 0.0001; Stacked CNN: 0.005 |

Please cite this article as: A. Ciuparu, A. Nagy-Dăbăcan and R.C. Mureșan, Soft++, a multi-parametric non-saturating non-linearity that improves convergence in deep neural architectures, Neurocomputing, https://doi.org/10.1016/j.neucom.2019.12.014
A stacked CNN architecture with only 11 layers, modified after the network in [8] and using fixed learning schedule and fixed dropout (see Methods). In addition, to have similar starting conditions we chose the parameters of Soft++ such that its gradient in the origin amounted to 1, thus matching the gradient of ELU. Given the weight initialization procedure, ELU and Soft++ networks started under the same conditions and diverged only as soon as the neurons’ total input deviated from 0. The major advantage of Soft++ over ELU is that by varying the relationship between c and k, the gradient in the origin can be flexibly set.

In addition, we also created two more network types based on SELU. First, we kept the same exact settings as for the other networks (ELU & Soft++), using the same weight initializer and same dropout scheme in order to have a one-to-one comparison with the other activation functions (SELU unadapted). Second, we adapted the SELU networks to operate in their preferred regime, using LeCun normal initialization and α-dropout (SELU), which was applied only to the last two layers.

When trained for 500 epochs, Softplus networks did not surpass chance level, a result consistent with their slow convergence observed on the previous tests. ReLU and PReLU networks did converge, but to a significantly lower performance than ELU and Soft++ (Fig. 4a). Indeed, ELU and Soft++ networks seemed to be in a different class than ReLU and PReLU. We therefore focused on these two networks, adding the two SELU variants as well, and performed the training for 1500 epochs (Fig. 4b). The unadapted SELU networks lost convergence after about 700 training epochs (Fig. 4b, left & middle), indicating that keeping the same exact conditions as for ELU and Soft++ is detrimental to their training. The adapted SELU, operating in its favorite conditions, performed better, finally reaching a similar performance to ELU. Soft++ networks significantly surpassed both ELU and SELU networks on CIFAR-100 (Fig. 4b, right; p < 0.05; heteroscedastic, double t-test) and they also had a tendency to overfit less (Fig. 4b, left).

We tested multiple parameter instantiations for Soft++ and found that increased performance is obtained with smaller values of c (i.e. c = 2, k = 1, in which case the gradient in the origin matches that of ELU), the best networks exceeding a performance of 70% on the test set (data not shown). Thus, the flexible adjustment of Soft++ parameters allows optimization of learning for very hard problems, like CIFAR-100.

### 3.4. Performance and learning curves on the visual cortex dataset

The MNIST, CIFAR-10 and CIFAR-100 datasets are well-known in the machine learning community and provide robust test cases for various machine learning frameworks because each class is relatively well represented, with hundreds / thousands of examples. However, in certain fields of science where machine learning is increasingly used to explore structure in the data, obtaining a large number of examples per class is difficult, if not impossible. One such case is neuroscience, where data is recorded from live animals and the measurement window is typically limited in time [36,41]. This leads to datasets with many features (huge amount of signals recorded) but with few classes (e.g., visual stimulus type) and few examples per class (tens of trials at best). To make things

---

Fig. 2. Performance of deep MLPs on the MNIST dataset. (a) Classification performance obtained after training the networks for 300 steps (last network retained) as a function of the dataset split set. Insets depict the performance of the best networks on the validation and test sets. (b) Number of training steps required to attain a performance of 50% on the training and validation sets, for each network type. Learning curves on the training (c) and validation set (d). Mean and s.e.m. shown.
worse, the scientist does not know which of the measured features is relevant and therefore includes a large array of features, many of which are not relevant. Examples may also be unbalanced across classes, introducing further complications [41]. Such datasets are found in many fields of science where measurement time is limited, they are typically small and noisy, and pose tremendous challenges to machine learning techniques. Learning from few examples on a large array of (possibly irrelevant) features is a very hard problem.

We next tested how deep MLPs performed on a dataset assembled from the firing statistics of 30 visual cortex neurons. Their responses were recorded using extra-cellular in vivo electrophysiology during visual stimulation with drifting gratings spanning 8 directions / 4 orientations (opposing directions merged: 0°–180°, 45°–225°, 90°–270°, 135°–315°; see Methods). These cortical neurons typically respond with elevated firing rates that are modulated by the passage of the grating bars through their receptive fields (Fig. 5a, top) and fire selectively to preferred directions/orientations (Fig. 5a, bottom). For a particular stimulus direction, these neurons not only fire selectively in their average firing rate but also exhibit a temporal modulation along the trial (Fig. 5b). Information about the particular visual stimulus presented to the animal can therefore be recovered by considering the relative time-resolved modulation of the firing of different neurons. By segmenting each trial into 10 bins and computing the number of spikes per each bin for each neuron, we assembled feature vec-

Fig. 3. Performance of the modified AlexNet (a–e) and stacked CNN (f–g) networks with various activation functions on the CIFAR-10 dataset. (a)–(c) Learning curves on the validation set over 100 epochs. The dotted line is the maximum performance achieved by Soft++ networks. (d) Performance at 15 training epochs. (e) Best performance on the test set after 100 epochs of training. In all cases k = 1. Learning curves of the 11-layered stacked CNN network using ELU, SELU and Soft++ on the train (f) and validation (g) sets of CIFAR-10. Mean and SD shown.
tors with 600 dimensions over 240 trials corresponding to the 4 grating orientations.

The visual cortex dataset was hard to learn for several reasons. First, cortical neurons fire with high trial-to-trial variability [44]. Second, the sampling of cortical activity with current in vivo technology is relatively random and minuscule, sampling just a few tens of units out of tens/hundreds of millions of neurons of the visual cortex. As a result, datasets extracted from such data are small and noisy. Indeed, ReLU and PReLU networks were not even able to learn on this dataset, remaining at chance level (Fig. 5c, 5e, and 5f). The same was the case for SELU networks (adapted) with dropout, which performed poorly on this dataset (data not shown).

PReLU networks could learn but poorer Fig. 5c, e, and f) and slower (Fig. 5d) than ELU and Soft++, which, again, seemed to be in a different league than the other activation functions. Overall, Soft++ performed robustly for different parameters but it had the best performance for a small $c$ (2), where it clearly surpassed ELU. Finally, Soft++ networks learned significantly faster than ELU networks (Fig. 5d; $p < 0.001$; heteroscedastic, double tailed t-test, Bonferroni corrected).

### 3.5. Performance and learning curves on the synthetic dataset

In a final set of tests we generated a synthetic dataset that represents an extreme case of a small, noisy dataset, with a majority of features irrelevant and only one relevant feature. This represents well many datasets found in neuroscience and other fields of science, whereby the scientist does not know a priori what feature is relevant and therefore includes all the empirically defined features. As expected, learning in fully-connected MLP networks was more difficult than on MNIST (Fig. 6). Softplus had the slowest initial convergence but reached a higher final performance than ReLU and SELU. The latter performed the worst...
Fig. 5. Predicting the visual stimulus presented to a rodent from the activity of visual cortex neurons. (a) Firing of an example cortical neuron (top) and its corresponding tuning-curve (bottom) in response to gratings drifting along directions covering the full circle, in steps of 45°. Each direction is shown 30 times, yielding 240 trials. (b) Spike raster (top) and peri–stimulus time-histogram (PSTH; bottom) of cortical neuron from (a) in response to direction 180°. (c) Classification performance for different sets, computed over the 4 grating drifting orientations and on the feature vectors extracted from the firing of the 30 cortical neurons in 240 trials. (d) Number of steps required by various networks to reach a performance of 50% on the training (left) and validation sets (right). ReLU and Softplus networks do not reach a performance of 50% and hence are not shown. Learning curves on the training (e) and validation set (f). Mean and s.e.m. shown.

on all sets (Fig. 6a), PReLU learned more aggressively than Softplus (Fig. 6b) but generalized worse than the latter (Fig. 6a, c, and d). Of all networks, ELU learned the most aggressively (Fig. 6b–d) but its validation and test set performances saturated below Soft++. The latter generalized significantly better, surpassing all other activation functions on the validation and test sets (Fig. 6a, c, d; compared to ELU, p < 0.01; heteroscedastic, double tailed t-test, Bonferroni corrected).

For the synthetic dataset, the explored slope coefficients of Soft++ did not have a significant effect on learning. However, increasing the exponent coefficient beyond 1 led to a rapid decrease in performance (data not shown). This suggests that the
slope of the activation function in the positive domain may have played an important role in enabling the network to learn the proper separatrix for the synthetic dataset.

4. Conclusions

We have investigated the importance of the shape of various activation functions in deep networks. Classical MLPs, with saturating activation functions (sigmoid, tanh), used to be very difficult to train due to the vanishing gradient problem [3]. Consequently, their structure was shallow (up to 1–3 hidden layers), to enable efficient training. By contrast, here we investigated networks with 6–11 hidden layers, which are considered deep [6,7]. Our results suggest that on such networks, both the non-saturating property and the actual shape of the activation function play a role in training. In addition, we also demonstrated that these conclusions hold in deep convolutional neural networks as well.

The most popular activation function for deep learning, ReLU, is non-saturating only in the positive domain. Our results indicate that it performed the worst on all tested datasets. Its fully non-saturating version, PReLU [15], performed better than ReLU but it was sometimes outperformed by Softplus, a saturating function in the negative domain, but smooth and non-linear around the origin. Of all activation functions, Softplus provided the slowest convergence. By contrast, ELU, another saturating function but with both positive and negative values around the origin, and with a significant gradient around it, is a remarkable activation function, providing in general fast and robust convergence, as reported in [8].

Its scaled variant, SELU [23] can also provide good results on some datasets but it seems less robust on very difficult or noisy datasets, and on datasets with many irrelevant features.

Here, we introduced Soft++, which is a novel activation function that extends both Softplus and PReLU, combining the advantages of both: it is smooth and non-linear around the origin and it is non-saturating, thus favoring training of deep networks. In addition, Soft++ has two parameters, which enable the flexible adjustment of the initial gradient for learning and of the slopes in the positive and negative domain. In general, networks endowed with this activation function outperformed those endowed with ReLU, PReLU, Softplus, SELU and ELU activation functions, converging significantly faster and/or reaching a higher classification performance on the validation and test sets, under the same conditions.

Smoothness and non-linearity around the origin appear to be even more important for learning datasets with a majority of irrelevant features. Networks having activation functions with purely linear slopes in the negative and/or positive domains (ReLU, PReLU) have a tendency to strongly overfit on such datasets. The smooth and nonlinear nature of Softplus and Soft++ around the origin avoids such overfitting. In addition, Soft++ networks behave surprisingly well on tasks with a high number of irrelevant features, discovering quickly the relevant ones. In such cases, Soft++ facilitates learning that also provides the conditions for robust generalization over the validation and test sets, minimizing cross-entropy.

Parametrization is an additional important property of PReLU, SELU and Soft++. PReLU and Soft++ activation functions can be...
adjusted to enable more or less aggressive learning: a smaller $c$ (larger $a$) corresponds to a steeper slope in the negative domain, while a larger $k$ determines a steeper slope in the positive domain. The double parameterization of Soft++ comes at a cost: the space of hyperparameters to be explored when training a Soft++ network is larger than the one of an identical network with a different activation function (e.g., RELU or ELU). Given a grid search on $M$ hyperparameters with $N$ values each, $N^M$ experiments are required to find the best configuration. In the case of Soft++, the number of hyperparameters is increased by two, hence $N^{M+2}$ experiments will be required.

Determining the best Soft++ parameters for a certain dataset is not always an easy task. On the other hand, following the approach in [15], the parameters of these functions can be learned and, even better, adjusted individually for each neuron. In a future study, it will be very interesting to address how Soft++ networks with learned $c$ and $k$ per each neuron fair on very difficult datasets, like CIFAR-100 or, e.g., the noisy and small datasets extracted from neuroscience data.

Soft++ outperformed the other activation functions on all the tested datasets. The function was initially developed to cope with the small and noisy datasets, with many irrelevant features, which are found in neuroscience, but its parametrization lends it a clear advantage also for the case of larger image datasets. By far, the strongest competitor of Soft++ is ELU, which performs remarkably well on all the tested problems. Soft++, ELU, and sometimes SELU appear to belong to a different league than all the other activation functions.

The most important message of the present study is that, for a given activation function, convergence speed and accuracy depend critically on the properties of the dataset and the architecture of the network. Therefore, one would ideally need a parametrized activation function that can be best adapted to the problem at hand. We have shown that Soft++ is such a versatile activation function, allowing one to obtain significantly faster and/or more accurate convergence than other activation functions. In particular, the flexibility of Soft++ and its smooth gradient over the entire real domain enables it to gain a critical advantage over ELU and SELU.

To conclude, combining the fully non-saturating character with a smooth and non-linear shape around the origin for the activation function is beneficial for training deep neural networks. Such a function is Soft++, whose dual parametrization enables the flexible choice of the slopes/gradients in the positive and negative domains, as well as the shape of the function around the origin. As a result, Soft++ can significantly accelerate learning and improve generalization on complex datasets, including those with many irrelevant features. It will be interesting to study how this activation function fares on difficult datasets once the slope coefficient, $c$, and exponent coefficient, $k$, are trained, individually for each neuron and along with the weights in the network.

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

The authors wish to thank Vasile V. Moa for constructive discussions and logistic support. This work was supported by: two grants from the Romanian National Authority for Scientific Research and Innovation, CNCS-UEFISCDI (project numbers PN-III-P4-ID-PCE-2016–0100, and COFUND-NEURON-NMDAR-PSV), a grant by the European Union’s Horizon 2020 research and innovation program – grant agreement no. 668683-Sybil-AAA, and a National Science Foundation grant NSF-IOS-1656830 funded by the US Government. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] H. Alnauilim, T.G. Dieterich, Learning boolean concepts in the presence of many irrelevant features, Artif. Intell. 69 (1–2) (1994) 297–305.
[2] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle, et al., Greedy layer-wise training of deep networks, Adv. Neural. Inf. Process. Syst. 19 (2007) 153.
[3] Y. Bengio, S.Simard, P. Frasconi, Learning long-term dependencies with gradient descent is difficult, IEEE Trans. Neural Netw. 5 (2) (1994) 154–166.
[4] M. Bianchini, F. Scarselli, On the complexity of neural network classifiers: a comparison between shallow and deep architectures, IEEE Trans. Neural Netw. Learn. Syst. 25 (8) (2014) 1553–1565.
[5] G. Chen, Deep learning with nonparametric clustering, arXiv:1505.03084 (2015).
[6] D. Cirea, U. Meier, J. Masci, J. Schmidhuber, Multi-column deep neural network for traffic sign classification, Neural Netw. 32 (2012) 333–338.
[7] D.C. Cirea, U. Meier, L.M. Gambardella, J. Schmidhuber, Big deep simple neural nets excel on handwritten digit recognition, Neural Comput. 22 (12) (2010) 3207–3220.
[8] D.-A. Clevert, T. Unterthiner, S. Hochreiter, Fast and accurate deep network learning by exponential linear units (ELUs), arXiv:1511.07289 (2015).
[9] C. Dugas, Y. Bengio, F. Bélisle, C. Nadeau, R Garcia, Incorporating second-order functional knowledge for better option pricing, Adv. Neural Inf. Process. Syst. (2001) 472–478.
[10] X. Glorot, Y. Bengio, Understanding the difficulty of training deep feedforward neural networks, Aistats 9 (2010) 249–256.
[11] P. Colak, P. Poetsch, H. Ney, Contrastive vs. squared error training: a theoretical and experimental comparison, Interspeech (2013) 1756–1760.
[12] I.J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, Y. Bengio, Maxout networks, Proceedings of the 30th International Conference on Machine Learning, PMLR 38 (2013) 1337–1345.
[13] H.A. Güvenir, A classification learning algorithm robust to irrelevant features, in: Proceedings of the International Conference on Artificial Intelligence: Methodology, Systems, and Applications, Springer, 1998, pp. 281–290.
[14] R.H. Hahnloser, R. Sarpeshkar, M.A. Mahowald, R.J. Douglas, H.S. Seung, Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit, Nature 405 (6789) (2000) 947–951.
[15] H. He, X. Zhang, S. Ren, J. Sun, Delving deep into rectifiers: surpassing human-level performance on imagenet classification, in: Proceedings of the IEEE International Conference on Computer Vision, 2015, pp. 1026–1034.
[16] G.E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, R.R. Salakhutdinv, Improving neural networks by preventing co-adaptation of feature detectors, arXiv:1207.0580 (2012).
[17] S Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions, Int. J. Uncert. Fuzz. Knowl. Based Syst. 6 (02) (1998) 107–116.
[18] D.H. HubeI, T.N. Wiesel, Receptive fields, binocular interaction and functional architecture in the cat’s visual cortex, J. Physiol (1962) 106–154.
[19] C. Igel, M. Husken, Improving the rprop learning algorithm, in: Proceedings of the Second International CSC Symposium on Neural Computation, 2000, Citeseer, 2000, pp. 115–121.
[20] C. Igel, M. Husken, Empirical evaluation of the improved RPROP learning algorithm, Neurocomputing (2003) 105–123.
[21] H. Ida, S. Shigei, H. Sato, Similarity of direction tuning among responses to stimulation of different whiskers in neurons of rat barrel cortex, J. Neurophysiol. (Sept. 2005) 2004–2018.
[22] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv:1412.6980 (2014).
[23] G. Klambauer, T. Unterthiner, A. Mayr, S. Hochreiter, Self-normalizing neural networks, in: Advances in Neural Information Processing Systems 30, Curran Associates, Inc, 2017, pp. 92–100.
[24] A. Krizhevsky, G.E. Hinton, Learning multiple layers of features from tiny images, Vol. 1. No. 4. Technical report, University of Toronto, 2009.
[25] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, Adv. Neural Inf. Process. Syst. (2012) 1097–1105.
[26] V. Kuklev, M. Sanguineti, Probabilistic lower bounds for approximation by shallow perceptron networks, Neural Netw. 91 (2017) 34–41.
[27] P Langley, Selection of relevant features in machine learning, in: Proceedings of the AAAI Fall Symposium on Relevance, 184, 1994, pp. 245–271.
[28] N. Le Roux, Y. Bengio, Representational power of restricted Boltzmann machines and deep belief networks, Neural Comput. 20 (6) (2008) 1631–1649.
[29] Y. LeCun, http://yann.lecun.com/exdb/mnist/.
[30] Y. LeCun, V. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444.
[31] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE 86 (11) (1998) 2278–2324.
[32] Y. LeCun, L. Bottou, G.B. Orr, K.-R. Müller, Efficient backprop, in: Neural Networks: Tricks of the Trade, Springer, 2012, pp. 9–48.
[33] M.S. Lewicki, A review of methods for spike sorting: the detection and classification of neural action potentials, Network 9 (Nov. 1998) R33–R78.
[34] A.L. Maas, A.Y. Hannun, A.Y. Ng, Rectifier nonlinearities improve neural network acoustic models, Proceeding of the ICML, 30, 2013.

[35] R.C. Mureșan, The coherence theory: simple attentional modulation effects, Neurocomputing 58-60 (2004) 949–955.

[36] E. Nedelcu, R. Portase, R. Tolas, R. Muresan, M. Dinsoareanu, R. Potolea, Artifact detection in eeg using machine learning, in: Proceedings of the 13th IEEE International Conference on Intelligent Computer Communication and Processing (ICCP), IEEE, 2017, pp. 77–83.

[37] A.Y. Ng, Feature selection, l 1 vs. l 2 regularization, and rotational invariance, in: Proceedings of the Twenty-First International Conference on Machine Learning, ACM, 2004, p. 78.

[38] S.-H. Oh, Improving the error backpropagation algorithm with a modified error function, IEEE Trans. Neural Netw. 8 (3) (1997) 799–803.

[39] M. Riedmiller, H. Braun, A direct adaptive method for faster backpropagation learning: the RPROP algorithm, in: Proceedings of the IEEE International Conference on Neural Networks, IEEE, 1993, pp. 586–591.

[40] D. Roy, K.S.R. Murty, C.K. Mohan, Feature selection using deep neural networks, in: Proceedings of the International Joint Conference on Neural Networks (IJCNN), IEEE, 2015, pp. 1–8.

[41] I.D. Rus, P. Marc, M. Dinsoareanu, R. Potolea, R.C. Mureșan, Classification of EEG signals in an object recognition task, in: Proceedings of the 13th IEEE International Conference Intelligent Computer Communication and Processing (ICCP), Sept. 2017, pp. 351–355.

[42] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al., Imagenet large scale visual recognition challenge, Int. J. Comput. Vis. 115 (3) (2015) 211–252.

[43] J. Schmidhuber, Deep learning in neural networks: an overview, Neural Netw. 61 (2015) 85–117.

[44] M.N. Shadlen, W.T. Newsome, The variable discharge of cortical neurons: implications for connectivity, computation, and information coding, J. Neurosci. Offic. J. Soc. Neurosci. 18 (May 1998) 3870–3896.

[45] D. Silver, A. Huang, C.J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al., Mastering the game of go with deep neural networks and tree search, Nature 529 (7587) (2016) 484–489.

Andrei Ciuparu received a Psychology degree from the Faculty of Psychology and Educational Sciences of Babeș-Bolyai University in Cluj-Napoca. He then completed the research-focused Cognitive Neuroscience Masters program at the Radboud University from Nijmegen on the Memory and Plasticity track. In his Masters thesis he studied neural correlates of memory consolidation in humans during sleep, using MEG and signal processing methods. He then sought a position in the Experimental and Theoretical Neuroscience Department of TINS, where he is working on developing new data analysis methods based on machine learning.

Adriana Nagy-Dăbâcan graduated from the Electrical Engineering Faculty at the Technical University of Cluj Napoca and specialized in Biomedical Engineering for her Masters in Trinity College Dublin and RWTH Aachen. She wrote her dissertation in Neuroscience at the University of Zurich, developing experimental setups for 2-photon imaging paradigms. With this background, she joined the Experimental and Theoretical Neuroscience Department for her PhD, where she studied brain circuits involved in visual perception by acquiring and analyzing electrophysiological signals. She is now a postdoctoral fellow at TINS, interested in the role of oscillatory circuit activity in visual processing.

Raul C. Mureșan received an engineering degree and then a PhD from the Faculty of Automation and Computer Science at the Technical University of Cluj-Napoca. During 2004–2008 he was a visiting PhD student and then a postdoc at the Frankfurt Institute for Advanced Studies and Max Planck Institute for Brain Research in Frankfurt, Germany. In 2008 he returned to Romania where he established his own lab that uses electroencephalography (EEG) on humans and optical imaging and electrophysiology in rodents to study how brain circuits give rise to perception and behavior. His lab also contributes to development of novel analysis methods and performs in silico simulations of neural architectures. In 2017 he co-founded the Transylvanian Institute of Neuroscience (TINS). He heads the Experimental and Theoretical Neuroscience Department and is the president of TINS.

Please cite this article as: A. Ciuparu, A. Nagy-Dăbâcan and R.C. Mureșan, Soft++, a multi-parametric non-saturating non-linearity that improves convergence in deep neural architectures, Neurocomputing, https://doi.org/10.1016/j.neucom.2019.12.014