UDC 004.8:004.032.26
DOI: 10.20535/SRIT.2308-8893.2022.1.07

ADAPTIVE HYBRID ACTIVATION FUNCTION FOR DEEP NEURAL NETWORKS

Ye. BODYANSKIY, S. KOSTIUK

Abstract. The adaptive hybrid activation function (AHAF) is proposed that combines the properties of the rectifier units and the squashing functions. The proposed function can be used as a drop-in replacement for ReLU, SiL and Swish activations for deep neural networks and can evolve to one of such functions during the training. The effectiveness of the function was evaluated on the image classification task using the Fashion-MNIST and CIFAR-10 datasets. The evaluation shows that the neural networks with AHAF activations achieve better classification accuracy comparing to their base implementations that use ReLU and SiL. A double-stage parameter tuning process for training the neural networks with AHAF is proposed. The proposed approach is sufficiently simple from the implementation standpoint and provides high performance for the neural network training process.

Keywords: adaptive hybrid activation function, double-stage parameter turning process, deep neural networks.

INTRODUCTION

In the recent years deep neural networks (DNNs) have got a wide proliferation for solving ranges of problems in virtually all areas of human activity, including the fields of Data Mining, Big Data, Data Science, digital video and audio signal processing, natural language processing, forecasting and control of complex systems [1–6].

The common property of all neural networks is their learning ability which consists of tuning the parameters (and, possibly, architectures) during the processing of available information and their universal approximation capabilities [7, 8] that allows to analyze and recover arbitrary complex nonlinear dependencies in the source data.

The most popular neural node of the DNN is the elementary perceptron of F. Rosenblatt which uses so-called squashing functions as their activation functions [7], such as sigmoid σ-functions, which are the most common squashing functions, tanh, Softsign, Satlin, aretan and others. At the same time the application of squashing functions runs against computational difficulties (so-called effect of vanishing gradient) when their derivatives approach zero while the input signal moves further from the origin.

Thereby instead of the squashing functions various DNN implementations commonly use piece-wise activation functions that belong to the so-called “rectified unit” family [9] which includes ReLU, ELU, PReLU, LReLU, NReLU and other similar functions [10–12]. It shall be noted that piece-wise activation functions allow only piece-wise approximation, i.e., the number of nodes and layers in the neural network shall be significantly increased to provide the required approximation capacity for non-trivial dependencies.

At the same time, there is a relatively wide group of recurrent neural networks [13] such as long-short-term memories, transformers and similar networks that use squashing functions in their gated recurrent units [3], so the hybrid activation functions were introduced that combine the properties of both the rectifiers and sigmoid functions. The list of hybrid functions includes [14], Swish [15], S-shaped [16], WiG [17] and other similar functions [18, 19].

All such hybrid activation functions have some free parameters that define their exact shape, amplitude and singular points which shall be in some way selected and adjusted for solving specific tasks. In this regard, it is advisable to introduce some additional procedures for automatic adjustment of the activation functions parameters. [20–25] address the off-line procedures that allow to find the required function parameters after the synaptic weights of the network are already set up. It is clear this approach significantly increases the training time.

In [26] the adaptive parametric rectified linear activation function (AdPReLU) was introduced where the parameters were adjusted simultaneously with the synaptic weights during the error backpropagation procedure. This approach allowed to reduce the training time and improve the quality of the obtained solution compared to Adaline, ReLU and tanh on the prediction task.

It is advisable to implement a similar approach for hybrid activation functions [14–19] and synthesize on their basis an adaptive activation function that is a generalization of the ones that are already used in the DNN applications.

ARCHITECTURE OF A NEURON WITH ADAPTIVE HYBRID ACTIVATION FUNCTION

Elementary perceptron of F. Rosenblatt as node of a neural network performs a non-linear transformation of the following form:

$$\hat{y}_j(k) = \psi_j\left(\theta_{j0} + \sum_{i=1}^{n} w_{ji}x_i(k)\right) = \psi_j\left(\sum_{i=0}^{n} w_{ji}x_i(k)\right) = \psi_j(w_j^T x(k)) = \psi_j(u_j(k)),$$

where $\hat{y}_j(k)$ — output signal of the j-th neuron of the network on the k-th data processing step, $k = 1,2,3,...,N,...$, $\psi_j(u_j(k))$ —non-linear transformation that is performed by the activation function on the signal of internal activation $u_j(k)$, θ_{j0} — threshold signal, w_{ji} — synaptic weight on the i-th input of the j-th neuron, $i = 0,1,2,...,n$, $w_{j0} \equiv \theta_{j0}$, $w_j = (w_{j0}, w_{j1},...,w_{jn})^T \in R^{n+1}$, $x(k) = (1, x_1(k),...$ $..., x_n(k))^T$ — $(n+1)\times 1$— dimensional vector of the input signals.

One of the most popular activation functions in the neural networks is a so-called sigmoid one that is studied by G. Cybenko [7] and has the following form:

$$\psi_j(u_j) = \sigma(\gamma_j u_j) = \frac{1}{1+e^{-\gamma_j u_j}}, \tag{1}$$

where γ_j — so-called gain parameter [20] that defines the shape of this function. The gain parameter value is often assumed to be equal to 1.

While the usage of sigmoid activation functions allows to provide universal approximation capabilities for the neural network, its application in DNNs runs up against computational complexities when the signal of internal activation starts to rise in its amplitude. In those cases, the derivative of the σ-function approaches zero, i.e., the effect of “vanishing gradient” increases.

To overcome this problem, we propose using a hybrid activation function of the following form:

$$\psi_j(u_j) = \beta_j u_j \sigma(\gamma_j u_j) = \frac{\beta_j u_j}{1+e^{-\gamma_j u_j}}, \tag{2}$$

where β_j and γ_j — parameters that shall be determined together with the synaptic weights during the training process. Being a modification of (1), activation function (2) does not suffer from the vanishing gradient effect. Note that the derivative of (2) by the signal of internal activation:

$$\frac{\partial \psi_j(u_j)}{\partial u_j} = \beta_j \sigma(\gamma_j u_j)(1 + u_j \gamma_j (1 - \sigma(\gamma_j u_j)))$$

produces small by amplitude values only when $u_j << 0$ that can be compensated by dialing the gain parameter γ_j.

Fig. 1 shows the architecture of an artificial neuron with adaptive hybrid activation function (2) (AHAF) in which function parameters β_j and γ_j are trained together with the vector of synaptic weights.

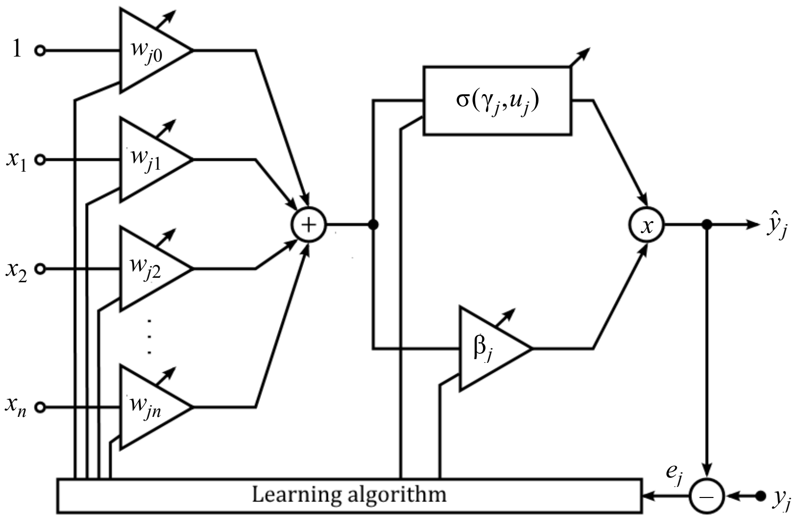

Fig. 1. Neuron with adaptive hybrid activation function (AHAF)

Here γ_j — external reference signal, $e_j = y_j - \hat{y}_j = y_0 - \psi_j(u_j) =$ $= y_j - \beta_j u_j (1 + e^{-\gamma_j u_j})^{-1}$ — learning error.

TRAINING ALGORITHM FOR A NEURON WITH AHAF

For training artificial neurons with AHAF we use the standard δ-rule [9] that for a regular perceptron of F. Rosenblatt and the error squared loss criteria:

$$E_j(k) = \frac{1}{2} e_j^2(k) = \frac{1}{2}(y_j(k) - \psi_j(u_j(k)))^2 = \frac{1}{2}\left(y_j(k) - \psi_j\left(\sum_{i=0}^{n} w_{ji} x_i(k) \right)\right)^2$$

allows to refine the synaptic weights with a recurrent procedure:

$$w_{ji}(k) = w_{ji}(k-1) - \eta_w(k)\frac{\partial E_j(k)}{\partial e_j(k)}\frac{\partial e_j(k)}{\partial w_{ji}} =$$

$$= w_{ji}(k-1) - \eta_w(k)e_j(k)\frac{\partial e_j(k)}{\partial w_{ji}} = w_{ji}(k-1) - \eta_w(k)e_j(k)\frac{\partial e_j(k)}{\partial u_j(k)}\frac{\partial u_j(k)}{\partial w_{ji}} =$$

$$= w_{ji}(k-1) + \eta_w(k)e_j(k)\psi'_j(u_j(k))x_i(k) = w_{ji}(k-1) + \eta_w(k)\delta_j(k)x_i(k),$$

where $\eta_w(k)$ — learning rate parameter the choice of which determines the convergence rate and the filtering (smoothing) abilities of the algorithm, $\delta_j(k) = e_j(k)\psi'_j(u_j(k))$ — so-called δ-error, based on which the error back-propagation procedure is implemented for training of multilayer neural networks.

For a neuron with AHAF that has a two-layer architecture (i.e., the first layer — synaptic weights $w_{ji} = 0,1,...,n$, the second — tunable parameters β_j and γ_j), backpropagation is implemented on a per-neuron level: parameters of the activation function are tuned first, then — the synaptic weights. This training procedure is referenced in this paper as the double-stage parameter tuning procedure (the DSPT procedure).

Considering that the δ-rule for tuning the activation function parameters $\psi_j(u_j,\beta_j,\gamma_j)$:

$$\frac{\partial\psi_j}{\partial\beta_j} = u_j\sigma(\gamma_j u_j) = \frac{u_j}{1+e^{-\gamma_j u_j}},$$

$$\frac{\partial\psi_j}{\partial\gamma_j} = \beta_j u_j^2\sigma(\gamma_j u_j)(1-\sigma(\gamma_j u_j)) = \beta_j\frac{u_j^2}{1+e^{-\gamma_j u_j}}\frac{e^{-\gamma_j u_j}}{1+e^{-\gamma_j u_j}}$$

can be written in the form of:

$$\beta_j(k) = \beta_j(k-1) - \eta_\beta(k)\frac{\partial E_j(k)}{\partial\beta_j} = \beta_j(k-1) + \eta_\beta(k)e_j(k)\frac{\partial\psi_j(k)}{\partial\beta_j} =$$

$$= \beta_j(k-1) + \eta_\beta(k)(y_j(k) - \psi_j(u_j(k),\beta_j(k-1),\gamma_j(k-1)))\times$$

$$\times u_j(k)\sigma(\gamma_j(k-1)u_j(k)),$$

where $u_j(k) = w_j^T(k-1)x(k)$, and:

$$\gamma_j(k) = \gamma_j(k-1) - \eta_\gamma(k)\frac{\partial E_j(k)}{\partial\gamma_j} = \gamma_j(k-1) + \eta_\gamma(k)e_j(k)\frac{\partial\psi_j(k)}{\partial\gamma_j} =$$

$$= \gamma_j(k-1) + \eta_\gamma(k)(y_j(k) - \psi_j(u_j(k),\beta_j(k-1),\gamma_j(k-1)))\times$$

$$\times\beta_j(k-1)u_j^2(k)\sigma(y_j(k-1)u_j(k))(1-\sigma(\gamma_j(k-1)u_j(k))) =$$

$$= \gamma_j(k-1) + \eta_\gamma(k)e_j(k)u_j^2(k)\sigma(y_j(k-1)u_j(k))(1-\sigma(\gamma_j(k-1)u_j(k))),$$

the training error can be recalculated after the tuning is performed for β_j and γ_j:

$$\tilde{e}_j(k) = y_j(k) - \psi_j(u_j(k),\beta_j(k),\gamma_j(k)) =$$

$$= y_j(k) - \frac{\beta_j(k)u_j(k)}{1+e^{-\gamma_j(k)u_j(k)}} = y_j(k) - \frac{\beta_j(k)w_j^{\mathrm{T}}(k-1)x(k)}{1+e^{-\gamma_j(k)w_j^{\mathrm{T}}(k-1)x(k)}},$$

and the synaptic weights are turned:

$$w_{ji}(k) = w_{ji}(k-1) + \eta_w(k)\tilde{e}_j(k)\frac{\partial\psi(u_j(k),\beta_j(k),\gamma_j(k))}{\partial u_j(k)}x_i(k) =$$

$$= w_{ji}(k-1) + \eta_w(k)\tilde{e}_j(k)\beta_j(k)\sigma(\gamma_j(k)u_j(k)) \times$$

$$\times(1+u_j(k)\gamma_j(k))(1-\sigma(\gamma_j(k)u_j(k)))x_i(k) = w_{ji}(k-1) + \eta_w(k)\tilde{\delta}_j(k)x_i(k),$$

where

$$\tilde{\delta}_j(k) = \tilde{e}_j(k)\psi'_j(u_j(k),\beta_j(k),\gamma_j(k)) =$$

$$= \tilde{e}_j(k)\beta_j(k)\sigma(\gamma_j(k)u_j(k))(1+u_j(k)\gamma_j(k)(1-\sigma(\gamma_j(k)u_j(k))).$$

With regards to selection of the learning rate parameters η_β, η_γ, η_w, the adaptive training algorithms like Adam [27], that are popular in DNNs, can be successfully replaced by the ones with the filtering and tracking properties [28] that have a sufficiently high speed of convergence.

For training of multi-layer networks, the hybrid error back propagation procedure can be used that, comparing to the standard one, calculates the training error and the δ-error twice per each hybrid layer of the network: $e_j(k)$, $\tilde{e}_j(k)$, $\delta_j(k)$, $\tilde{\delta}_j(k)$.

EVALUATION

Performance of the adaptive hybrid activation function was evaluated on the image classification task on two different datasets with two base neural network architectures in a similar way to [29]. The base architectures were modified to use AHAF activations instead of "classic" activations like ReLU and SiL. The performance of the modified networks was compared to the reference implementations. The neural network implementations together with the valuation and training environment were coded in Python 3.8 using PyTorch 1.9.0 [30]. The implementation is publicly available on GitHub: https://git.io/JDBIZ.

A. Dataset

The models with adaptive hybrid activation function were evaluated on two datasets: Fashion-MNIST [31] and CIFAR-10 [32].

Fashion-MNIST is a dataset that contains 60000 monochrome images, each 28×28 pixels in size, with associated class labels. Out of all images, 50000 images are used for training and 10000 are used for validation. The classes are exclusive, the one-hot encoding was used for the class labels. The pixel values were divided by 255 to rescale them to the $[0,0\cdots1,0]$ range. The images were augmented using the random horizontal flip with the flip probability of 0,5 and the random shift by both width and height with the maximum shift factor of 0,1.

CIFAR-10 is a dataset of 60000 RGB images, each 32×32 pixels in size and each having a one of 10 class labels associated with it. The train to test distribution is 5:1, where all images are randomly selected from the whole dataset. The classes are exclusive, the one-hot encoding was used for the class labels. Pixel values on all color channels were rescaled to the [0,0..1,0] range using division by 255. The training set was augmented using the random horizontal flip with probability of 0,5 and the random horizontal and vertical shift by the maximum factor of 0,1.

B. Neural Networks and Activations

Two base neural networks architectures were used in the experiment: LeNet-5 [33] and KerasNet from Keras version 1.2.2 [34].

LeNet-5 is a simple convolutional neural network consisting of 4 layers: 2 convolutional layers with pooling and activation functions, 1 linear layer with an activation function and 1 output linear layer with Softmax. The convolutional layers use 5×5 filters with 20 output channels for the first layer and 50 output channels for the second layer. Max pooling with the kernel of 2×2 is used as the pooling implementation. The hidden linear layer has 500 output features, the output layer has 10, one per each class. Several variants of LeNet-5 were used for evaluation: one with ReLU activations for the hidden layers, one with SiL, one AHAF activation initialized as ReLU and one with AHAF activation initialized as SiL. The total number of parameters depends on the size of the input images: 431000 and 657000 for Fashion-MNIST and CIFAR-10 correspondingly. The total number of parameters does not count the parameters of AHAF activations.

KerasNet is a neural network that is partially similar to VGG. The network has 6 layers: 4 convolutional layers with activation functions with each second layer followed by max pooling with dropout, 1 hidden linear layer with an activation function and dropout, 1 output linear layer with Softmax activation. The first and the second convolutional layers have 32 output channels with 3×3 filters, the first layer applies 1×1 padding to its input, while the second one does not apply any padding. Max pooling with 2×2 kernels and the dropout with the probability of 0,25 follow the first two convolutional layers. The third and the fourth convolutional layers use 3×3 filters and have 64 output channels, the third layer applies 1×1 padding while the fourth does not apply any padding. Max pooling with the kernel size of 2×2 and the dropout with the probability of 0,25 are used after the third and fourth convolutional layers. The hidden linear layer has 512 output features, dropout with the probability of 0,5 is applied after the hidden linear layer. The output layer has 10 output features, one per each class. Several variants of KerasNet were used for evaluation: one with ReLU activations for the hidden layers, one with SiL, one AHAF activation initialized as ReLU and one with AHAF activation initialized as SiL. The total number of parameters depends on the size of the input images: 889834 and 1250858 for Fashion-MNIST and CIFAR-10 correspondingly. The total number of parameters does not count the parameters of AHAF activations.

C. Training Procedures

The neural networks were trained on the Fashion-MNIST and CIFAR-10 datasets with the batch size of 64 for 100 epochs on a laptop with NVIDIA GeForce GTX 1650 Max-Q. The RMSprop optimizer was used for training with the initial learning rate of 10^{-4} and the learning rate decay of 10^{-6} applied per one minibatch.

The neural network variants with AHAF activations were trained using the "classic" training procedure (when all trainable parameters are updated in one go) and the DSPT procedure. Implementation of the DSPT procedure uses separate instances of the optimizer class per each set of parameters one per all AHAF parameters, one per the trainable parameters outside of AHAF activations.

The training set loss and the test set accuracy were recorded per for each of the training runs. The results of the training are analyzed and presented in the following section.

D. Analysis of Results

The network variants with AHAF activations outperform the base implementations with ReLU and SiL activations on both CIFAR-10 and Fashion-MNIST. LeNet-5 achieves the best results on the Fashion-MNIST dataset with AHAF activations initialized as ReLU and the DSPT procedure. KerasNet achieves the best results on the CIFAR-10 dataset with AHAF activations initialized as SiL and the DSPT procedure. Table presents the best achieved test set accuracy and the epoch number when this result was achieved for each of the network variants, datasets and parameter tuning procedures used for evaluation.

Best test set accuracy, up to 100 epochs

Network	Activ.	Init.	Proc.	Fashion-MNIST		CIFAR-10	
				Acc.,%	Epoch	Acc.,%	Epoch
LeNet-5	ReLU	N/A	Classic	91,43	98	75,89	96
LeNet-5	SiL	N/A	Classic	90,60	95	73,76	95
LeNet-5	AHAF	ReLU	Classic	91,55	99	76,69	95
LeNet-5	AHAF	SiL	Classic	91,16	99	74,47	99
LeNet-5	AHAF	ReLU	DSPT	91,73	93	74,44	95
LeNet-5	AHAF	SiL	DSPT	90,95	100	74,05	95
KerasNet	ReLU	N/A	Classic	91,29	100	79,36	97
KerasNet	SiL	N/A	Classic	91,76	93	79,83	99
KerasNet	AHAF	ReLU	Classic	91,30	84	79,71	100
KerasNet	AHAF	SiL	Classic	92,02	97	80,31	98
KerasNet	AHAF	ReLU	DSPT	91,35	55	79,30	96
KerasNet	AHAF	SiL	DSPT	91,96	98	80,37	98

Analysis of the dependency between the training loss, test set accuracy and the training epoch shows the potential for performance improvements using longer training runs (running the training for more epochs), different optimizers and learning rates. For KerasNet on the CIFAR-10 dataset the SiL-initialized AHAF activation function consistently shows lower training loss and higher test set accuracy comparing to the base implementation with SiL. Fig. 2 illustrates the

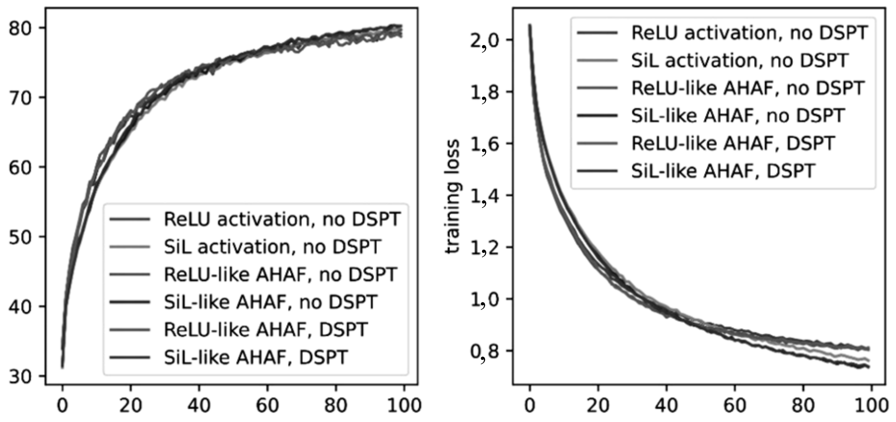

Fig. 2. Dependency between the loss, accuracy and the training epoch for KerasNet network on CIFAR-10

dependency between the training loss, the test set error, and the training epoch for the KerasNet network trained on the CIFAR-10 dataset.

For neural networks with AHAF initialized as ReLU, AHAF keeps its ReLU-like form, but changes the amplitude during the training process. This observation can be explained by the values of the gradient with respect to the γ parameter — the gradient decreases with the increase of the γ parameter. For neural networks with AHAF initialized as SiL, AHAF changes its form and amplitude during the training process. Fig. 3 and Fig. 4 show the form of the activation functions for the two final neurons of the KerasNet network trained on the CIFAR-10 dataset with ReLU-like and SiL-like AHAF activations correspondingly.

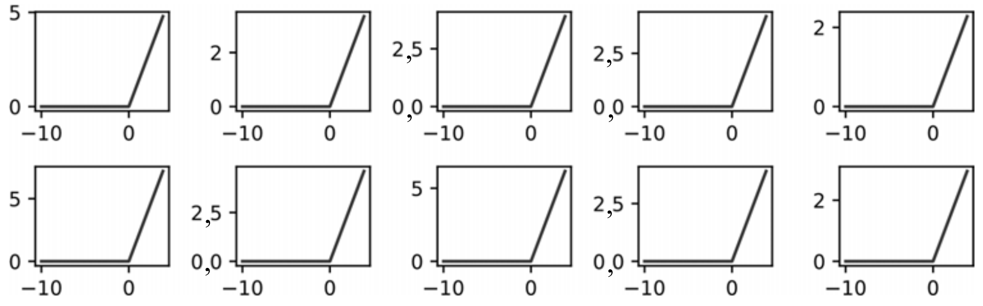

Fig. 3. The activation function form for AHAF initialized as ReLU

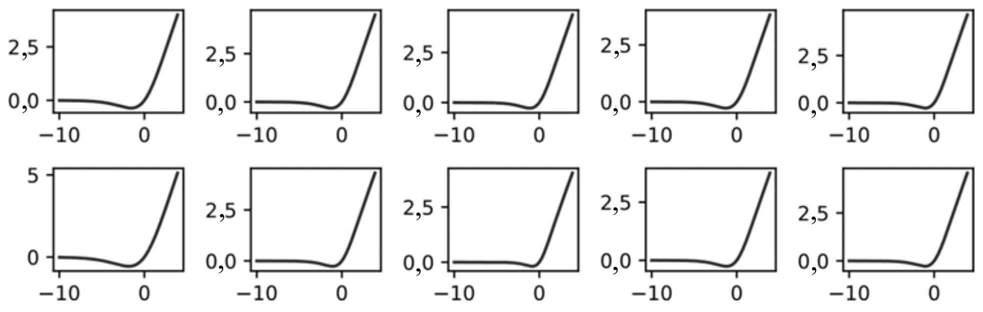

Fig. 4. The activation function form for AHAF initialized as SiL

CONCLUSIONS

Proposed an adaptive hybrid activation function (AHAF) that is applicable for usage in feed-forward and recurrent deep neural networks and combines the properties of both squashing functions and the ones from the rectified unit family. This function does not suffer from the effect of "vanishing gradient" and its parameters are trained together with the synaptic weights. Introduced a training algorithm for a neuron based on AHAF.

The proposed approach is sufficiently simple from the implementation standpoint and provides high performance for the neural network training process.

REFERENCES

1. Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning", *Nature*, vol. 521, no. 7553, pp. 436–444, 2015. doi: 10.1038/nature14539.
2. J. Schmidhuber, "Deep learning in neural networks: An overview", *Neural Networks*, vol. 61, pp. 85–117, 2015. doi: 10.1016/j.neunet.2014.09.003.

3. I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.
4. D. Graupe, *Deep Learning Neural Networks: Design and Case Studies.* USA: World Scientific Publishing Co., Inc., 2016.
5. A.L. Caterini and D.E. Chang, *Deep Neural Networks in a Mathematical Framework*, 1st ed. Springer Publishing Company, Incorporated, 2018.
6. C.C. Aggarwal, *Neural Networks and Deep Learning: A Textbook*, 1st ed. Springer Publishing Company, Incorporated, 2018.
7. G. Cybenko, "Approximation by superpositions of a sigmoidal function", *Mathematics of Control, Signals and Systems*, vol. 2, no. 4, pp. 303–314, 1989. doi: 10.1007/BF02551274.
8. K. Hornik, "Approximation capabilities of multilayer feedforward networks", *Neural Networks*, vol. 4, no. 2, pp. 251–257, 1991. doi: 10.1016/0893-6080(91)90009-T.
9. A. Cichocki and R. Unbehauen, *Neural Networks for Optimization and Signal Processing*, 1st ed. USA: John Wiley & Sons, Inc., 1993.
10. K. He, X. Zhang, S. Ren, and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification", in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, pp. 1026–1034. doi: 10.1109/ICCV.2015.123.
11. D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)", *arXiv [cs.LG]*, 2016. doi: 10.1162/neco.1997.9.8.1735.
12. K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition", in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778. doi: 10.1109/CVPR.2016.90.
13. S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory", *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997. doi: 10.1162/neco.1997.9.8.1735.
14. S. Elfwing, E. Uchibe, and K. Doya, "Sigmoid-Weighted Linear Units for Neural Network Function Approximation in Reinforcement Learning", *arXiv [cs.LG]*, 2017.
15. P. Ramachandran, B. Zoph, and Q.V. Le, "Searching for Activation Functions", *arXiv [cs.NE]*, 2017.
16. X. Jin, C. Xu, J. Feng, Y. Wei, J. Xiong, and S. Yan, "Deep Learning with S-shaped Rectified Linear Activation Units", *arXiv [cs.CV]*, 2015.
17. M. Tanaka, "Weighted Sigmoid Gate Unit for an Activation Function of Deep Neural Network", *arXiv [cs.CV]*, 2018.
18. B. Yuen, M.T. Hoang, X. Dong, and T. Lu, "Universal Activation Function For Machine Learning", *arXiv [cs.LG]*, 2020.
19. D. Misra, "Mish: A Self Regularized Non-Monotonic Activation Function", *arXiv [cs.LG]*, 2020.
20. J.K. Kruschke and J.R. Movellan, "Benefits of gain: speeded learning and minimal hidden layers in back-propagation networks", *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 21, no. 1, pp. 273–280, 1991. doi: 10.1109/21.101159.
21. Z. Hu and H. Shao, "The study of neural network adaptive control systems", *Control and Decision*, no. 7, pp. 361–366, 1992.
22. C.-T. Chen and W.-D. Chang, "A Feedforward Neural Network with Function Shape Autotuning", *Neural Netw.*, vol. 9, no. 4, pp. 627–641, 1996. doi: 10.1016/0893-6080(96)00006-8.
23. E. Trentin, "Networks with Trainable Amplitude of Activation Functions", *Neural Netw.*, vol. 14, no. 4–5, pp. 471–493, 2001. doi: 10.1016/S0893-6080(01)00028-4.
24. F. Agostinelli, M. Hoffman, P. Sadowski, and P. Baldi, "Learning Activation Functions to Improve Deep Neural Networks", *arXiv [cs.NE]*, 2015.
25. L.R. Sütfeld, F. Brieger, H. Finger, S. Füllhase, and G. Pipa, "Adaptive Blending Units: Trainable Activation Functions for Deep Neural Networks", *arXiv [cs.LG]*, 2018.
26. Y.V. Bodyanskiy, A. Deineko, I. Pliss, and V. Slepanska, "Formal Neuron Based on Adaptive Parametric Rectified Linear Activation Function and its Learning", in *Proc. 1st Int. Workshop on Digital Content & Smart Multimedia "DCSMART 2019"*, vol. 2533, pp. 14–22.
27. D.P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization", *arXiv [cs.LG]*, 2017.

28. P. Otto, Y. Bodyanskiy, and V. Kolodyazhniy, "A new learning algorithm for a forecasting neuro-fuzzy network", *Integrated Computer-Aided Engineering*, vol. 10, pp. 399–409, 2003. doi: 10.3233/ICA-2003-10409.
29. F. Manessi and A. Rozza, "Learning Combinations of Activation Functions", *CoRR*, vol. abs/1801.09403, 2018.
30. A. Paszke et al., "PyTorch: An Imperative Style, High-Performance Deep Learning Library", in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Reds Curran Associates, Inc., 2019, pp. 8024–8035.
31. H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms", *arXiv [cs.LG]*, 2017.
32. A. Krizhevsky, *Learning multiple layers of features from tiny images*, 2009.
33. Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition", *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. doi: 10.1109/5.726791.
34. F. Chollet et al., "Keras", 2015. [Online]. Available: https://github.com/ fchollet/keras.

Received 17.12.2021

INFORMATION ON THE ARTICLE

Yevgeniy V. Bodyanskiy, ORCID: 0000-0001-5418-2143, Kharkiv National University of Radio Electronics, Ukraine, e-mail: yevgeniy.bodyanskiy@nure.ua

Serhii O. Kostiuk, ORCID: 0000-0003-4196-2524, Kharkiv National University of Radio Electronics, Ukraine, e-mail: serhii.kostiuk@nure.ua

АДАПТИВНА ГІБРИДНА ФУНКЦІЯ АКТИВАЦІЇ ДЛЯ ГЛИБОКИХ НЕЙРОННИХ МЕРЕЖ / Є.В. Бодянський, С.О. Костюк

Анотація. Запропоновано адаптивну гібридну функцію активації (AHAF), що поєднує особливості випрямних блоків (rectifier units) та стискальних (squashing) функцій. Запропонована функція може бути використана як пряма заміна активаційних функцій ReLU, SiL і Swish для глибоких нейронних мереж, а також набути форми однієї з цих функцій в процесі навчання. Ефективність функції досліджено на задачі класифікації зображень на наборах даних Fashion-MNIST і CIFAR-10. Результати дослідження показують, що нейронні мережі з активаційними функціями AHAF показують точність класифікації кращу, ніж їх базові реалізації на основі ReLU та SiL. Запропоновано двоетапний процес налаштування параметрів для навчання нейронних мереж з AHAF. Запропонований підхід достатньо простий в реалізації та забезпечує високу продуктивність у навчанні нейронної мережі.

Ключові слова: адаптивна гібридна функція активації, двоетапний процес налаштування параметрів, глибокі нейронні мережі.

АДАПТИВНАЯ ГИБРИДНАЯ ФУНКЦИЯ АКТИВАЦИИ ДЛЯ ГЛУБОКИХ НЕЙРОННЫХ СЕТЕЙ / Е.В. Бодянский, С.А. Костюк

Аннотация. Предложена адаптивная гибридная функция активации (AHAF), которая объединяет свойства выпрямительных блоков (rectifier units) и сжимающих (squashing) функций. Предложенная функция может быть использована как прямая замена активационных функций ReLU, SiL и Swish для глубоких нейронных сетей, а также принимать форму одной из этих функций в процессе обучения. Эффективность функции исследована на задаче классификации изображений на наборах данных Fashion-MNIST и CIFAR-10. Результаты исследования показывают, что нейронные сети с активационными функциями AHAF показывают точность классификации лучшую, чем их базовые реализации на основе ReLU и SiL. Предложено двухэтапный процесс настройки параметров для обучения нейронных сетей с AHAF. Предложенный подход достаточно простой в реализации и обеспечивает высокую продуктивность в обучении нейронной сети.

Ключевые слова: адаптивная гибридная функция активации, двухэтапный процесс настройки параметров, глубокие нейронные сети.