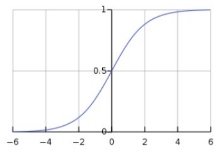
WIKIPEDIA

Activation function

In artificial neural networks, the **activation function** of a node defines the output of that node, or "neuron," given an input or set of inputs. This output is then used as input for the next node and so on until a desired solution to the original problem is found.[1]



Logistic activation function

It maps the resulting values into the desired range such as between 0 to 1 or -1 to 1 etc. (depending upon the choice of activation function). For example, the use of the logistic activation function would map all inputs in the real number domain into the range of 0 to 1.

A standard computer chip circuit can be seen as a digital network of activation functions that can be "ON" (1) or "OFF" (0), depending on input. This is similar to the behavior of the linear perceptron in neural networks. However, only *nonlinear* activation functions allow such networks to compute nontrivial problems using only a small number of nodes.[2] In artificial neural networks, this function is also called the transfer function.

Contents

Functions

In biologically inspired neural networks, the activation function is usually an abstraction representing the rate of action potential firing in the cell.[3] In its simplest form, this function is binary—that is, either the neuron is firing or not. The function looks like $\phi(v_i) = U(v_i)$, where U is the Heaviside step function. In this case many neurons must be used in computation beyond linear separation of categories.

A line of positive slope may be used to reflect the increase in firing rate that occurs as input current increases. Such a function would be of the form $\phi(v_i) = \mu v_i$, where μ is the slope. This activation function is linear, and therefore has the same problems as the binary function. In addition, networks constructed using this model have unstable convergence because neuron inputs along favored paths tend to increase without bound, as this function is not normalizable.

All problems mentioned above can be handled by using a normalizable sigmoid activation function. One realistic model stays at zero until input current is received, at which point the firing frequency increases quickly at first, but gradually approaches an asymptote at 100% firing rate. Mathematically, this looks like $\phi(v_i) = U(v_i)\tanh(v_i)$, where the hyperbolic tangent function can be replaced by any sigmoid function. This behavior is realistically reflected in the neuron, as neurons cannot physically fire faster than a certain rate. This model runs into problems, however, in computational networks as it is not differentiable, a requirement to calculate backpropagation.

The final model, then, that is used in multilayer perceptrons is a sigmoidal activation function in the form of a hyperbolic tangent. Two forms of this function are commonly used: $\phi(v_i) = \tanh(v_i)$ whose range is normalized from -1 to 1, and $\phi(v_i) = (1 + \exp(-v_i))^{-1}$ is vertically translated to normalize from 0 to 1. The latter model is often considered more biologically realistic, but it runs into theoretical and experimental difficulties with certain types of computational problems.

Alternative structures

A special class of activation functions known as radial basis functions (RBFs) are used in RBF networks, which are extremely efficient as universal function approximators. These activation functions can take many forms, but they are usually found as one of three functions:

- Gaussian: $\phi(v_i) = \exp\left(-\frac{\|v_i - c_i\|^2}{2\sigma^2}\right)$
- Multiquadratics: $\phi(v_i) = \sqrt{\|v_i - c_i\|^2 + a^2}$
- Inverse multiquadratics: $\phi(v_i) = (\|v_i - c_i\|^2 + a^2)^{-1/2}$

where c_i is the vector representing the function *center* and a and σ are parameters affecting the spread of the radius.

Support vector machines (SVMs) can effectively utilize a class of activation functions that includes both sigmoids and RBFs. In this case, the input is transformed to reflect a decision boundary hyperplane based on a few training inputs called *support vectors* x. The activation function for the hidden layer of these machines is referred to as the *inner product kernel*, $K(v_i, x) = \phi(v_i)$. The support vectors are represented as the centers in RBFs with the kernel equal to the activation function, but they take a unique form in the perceptron as

$$\phi(v_i) = \tanh\left(\beta_1 + \beta_0 \sum_j v_{i,j} x_j\right),$$

where β_0 and β_1 must satisfy certain conditions for convergence. These machines can also accept arbitrary-order polynomial activation functions where

$$\phi(v_i) = \left(1 + \sum_j v_{i,j} x_j\right)^p.$$ [4]

Activation function having types:

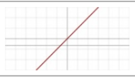
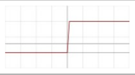
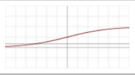
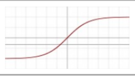
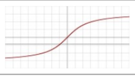
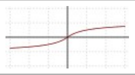
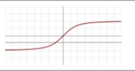
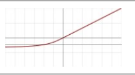
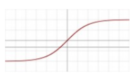
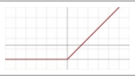
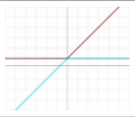
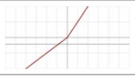
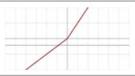
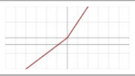
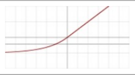
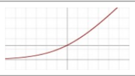
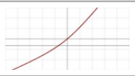
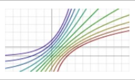
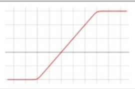
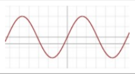
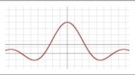
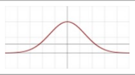
- Identity function
- Binary step function
- Bipolar step function
- Sigmoidal function
  - Binary sigmoidal function
  - Bipolar sigmoidal function
- Ramp function

Comparison of activation functions

Some desirable properties in an activation function include:

- Nonlinear – When the activation function is non-linear, then a two-layer neural network can be proven to be a universal function approximator.[5] The identity activation function does not satisfy this property. When multiple layers use the identity activation function, the entire network is equivalent to a single-layer model.
- Range – When the range of the activation function is finite, gradient-based training methods tend to be more stable, because pattern presentations significantly affect only limited weights. When the range is infinite, training is generally more efficient because pattern presentations significantly affect most of the weights. In the latter case, smaller learning rates are typically necessary.
- Continuously differentiable – This property is desirable (RELU is not continuously differentiable and has some issues with gradient-based optimization, but it is still possible) for enabling gradient-based optimization methods. The binary step activation function is not differentiable at 0, and it differentiates to 0 for all other values, so gradient-based methods can make no progress with it.[6]
- Monotonic – When the activation function is monotonic, the error surface associated with a single-layer model is guaranteed to be convex.[7]
- Smooth functions with a monotonic derivative – These have been shown to generalize better in some cases.
- Approximates identity near the origin – When activation functions have this property, the neural network will learn efficiently when its weights are initialized with small random values. When the activation function does not approximate identity near the origin, special care must be used when initializing the weights.[8] In the table below, activation functions where $f(0) = 0$ and $f'(0) = 1$ and f' is continuous at 0 are indicated as having this property.

The following table compares the properties of several activation functions that are functions of one fold x from the previous layer or layers:

Name	Plot	Equation	Derivative (with respect to *x*)	Range	Order of continuity	Monotonic	Monotonic derivative	Approximates identity near the origin
Identity		$f(x)=x$	$f'(x)=1$	$(-\infty,\infty)$	C^∞	Yes	Yes	Yes
Binary step		$f(x)=\begin{cases}0 & \text{for } x<0\\ 1 & \text{for } x\geq 0\end{cases}$	$f'(x)=\begin{cases}0 & \text{for } x\neq 0\\ ? & \text{for } x=0\end{cases}$	$\{0,1\}$	C^{-1}	Yes	No	No
Logistic (a.k.a. Sigmoid or Soft step)		$f(x)=\sigma(x)=\frac{1}{1+e^{-x}}$[1]	$f'(x)=f(x)(1-f(x))$	$(0,1)$	C^∞	Yes	No	No
TanH		$f(x)=\tanh(x)=\frac{(e^x-e^{-x})}{(e^x+e^{-x})}$	$f'(x)=1-f(x)^2$	$(-1,1)$	C^∞	Yes	No	Yes
ArcTan		$f(x)=\tan^{-1}(x)$	$f'(x)=\frac{1}{x^2+1}$	$\left(-\frac{\pi}{2},\frac{\pi}{2}\right)$	C^∞	Yes	No	Yes
ElliotSig[9][10][11] Softsign[12][13]		$f(x)=\frac{x}{1+\lvert x\rvert}$	$f'(x)=\frac{1}{(1+\lvert x\rvert)^2}$	$(-1,1)$	C^1	Yes	No	Yes
Inverse square root unit (ISRU)[14]		$f(x)=\frac{x}{\sqrt{1+\alpha x^2}}$	$f'(x)=\left(\frac{1}{\sqrt{1+\alpha x^2}}\right)^3$	$\left(-\frac{1}{\sqrt{\alpha}},\frac{1}{\sqrt{\alpha}}\right)$	C^∞	Yes	No	Yes
Inverse square root linear unit (ISRLU)[14]		$f(x)=\begin{cases}\frac{x}{\sqrt{1+\alpha x^2}} & \text{for } x<0\\ x & \text{for } x\geq 0\end{cases}$	$f'(x)=\begin{cases}\left(\frac{1}{\sqrt{1+\alpha x^2}}\right)^3 & \text{for } x<0\\ 1 & \text{for } x\geq 0\end{cases}$	$\left(-\frac{1}{\sqrt{\alpha}},\infty\right)$	C^2	Yes	Yes	Yes
Square Nonlinearity (SQNL)[11]		$f(x)=\begin{cases}1 & : x>2.0\\ x-\frac{x^2}{4} & : 0\leq x\leq 2.0\\ x+\frac{x^2}{4} & : -2.0\leq x<0\\ -1 & : x<-2.0\end{cases}$	$f'(x)=1\mp\frac{x}{2}$	$(-1,1)$	C^∞	Yes	No	Yes
Rectified linear unit (ReLU)[15]		$f(x)=\begin{cases}0 & \text{for } x<0\\ x & \text{for } x\geq 0\end{cases}$	$f'(x)=\begin{cases}0 & \text{for } x<0\\ 1 & \text{for } x\geq 0\end{cases}$	$[0,\infty)$	C^0	Yes	Yes	No
Bipolar rectified linear unit (BReLU)[16]		$f(x_i)=\begin{cases}ReLU(x_i) & \text{if } i \bmod 2=0\\ -ReLU(-x_i) & \text{if } i \bmod 2\neq 0\end{cases}$	$f'(x_i)=\begin{cases}ReLU'(x_i) & \text{if } i \bmod 2=0\\ -ReLU'(-x_i) & \text{if } i \bmod 2\neq 0\end{cases}$	$(-\infty,\infty)$	C^0	Yes	Yes	No
Leaky rectified linear unit (Leaky ReLU)[17]		$f(x)=\begin{cases}0.01x & \text{for } x<0\\ x & \text{for } x\geq 0\end{cases}$	$f'(x)=\begin{cases}0.01 & \text{for } x<0\\ 1 & \text{for } x\geq 0\end{cases}$	$(-\infty,\infty)$	C^0	Yes	Yes	No
Parameteric rectified linear unit (PReLU)[18]		$f(\alpha,x)=\begin{cases}\alpha x & \text{for } x<0\\ x & \text{for } x\geq 0\end{cases}$	$f'(\alpha,x)=\begin{cases}\alpha & \text{for } x<0\\ 1 & \text{for } x\geq 0\end{cases}$	$(-\infty,\infty)$[2]	C^0	Yes iff $\alpha\geq 0$	Yes	Yes iff $\alpha=1$
Randomized leaky rectified linear unit (RReLU)[19]		$f(\alpha,x)=\begin{cases}\alpha x & \text{for } x<0\\ x & \text{for } x\geq 0\end{cases}$[3]	$f'(\alpha,x)=\begin{cases}\alpha & \text{for } x<0\\ 1 & \text{for } x\geq 0\end{cases}$	$(-\infty,\infty)$	C^0	Yes	Yes	No
Exponential linear unit (ELU)[20]		$f(\alpha,x)=\begin{cases}\alpha(e^x-1) & \text{for } x\leq 0\\ x & \text{for } x>0\end{cases}$	$f'(\alpha,x)=\begin{cases}f(\alpha,x)+\alpha & \text{for } x\leq 0\\ 1 & \text{for } x>0\end{cases}$	$(-\alpha,\infty)$	$\begin{cases}C_1 & \text{when } \alpha=1\\ C_0 & \text{otherwise}\end{cases}$	Yes iff $\alpha\geq 0$	Yes iff $0\leq\alpha\leq 1$	Yes iff $\alpha=1$
Scaled exponential linear unit (SELU)[21]		$f(\alpha,x)=\lambda\begin{cases}\alpha(e^x-1) & \text{for } x<0\\ x & \text{for } x\geq 0\end{cases}$ with $\lambda=1.0507$ and $\alpha=1.67326$	$f'(\alpha,x)=\lambda\begin{cases}\alpha(e^x) & \text{for } x<0\\ 1 & \text{for } x\geq 0\end{cases}$	$(-\lambda\alpha,\infty)$	C^0	Yes	No	No
S-shaped rectified linear activation unit (SReLU)[22]		$f_{t_l,a_l,t_r,a_r}(x)=\begin{cases}t_l+a_l(x-t_l) & \text{for } x\leq t_l\\ x & \text{for } t_l<x<t_r\\ t_r+a_r(x-t_r) & \text{for } x\geq t_r\end{cases}$ t_l,a_l,t_r,a_r are parameters.	$f'_{t_l,a_l,t_r,a_r}(x)=\begin{cases}a_l & \text{for } x\leq t_l\\ 1 & \text{for } t_l<x<t_r\\ a_r & \text{for } x\geq t_r\end{cases}$	$(-\infty,\infty)$	C^0	No	No	No
Adaptive piecewise linear (APL)[23]		$f(x)=\max(0,x)+\sum_{s=1}^{S}a_i^s\max(0,-x+b_i^s)$	$f'(x)=H(x)-\sum_{s=1}^{S}a_i^sH(-x+b_i^s)$[4]	$(-\infty,\infty)$	C^0	No	No	No
SoftPlus[24]		$f(x)=\ln(1+e^x)$	$f'(x)=\frac{1}{1+e^{-x}}$	$(0,\infty)$	C^∞	Yes	Yes	No
Bent identity		$f(x)=\frac{\sqrt{x^2+1}-1}{2}+x$	$f'(x)=\frac{x}{2\sqrt{x^2+1}}+1$	$(-\infty,\infty)$	C^∞	Yes	Yes	Yes
Sigmoid Linear Unit (SiLU)[25] (AKA SiL[26] and Swish-1[27])		$f(x)=x\cdot\sigma(x)$[5]	$f'(x)=f(x)+\sigma(x)(1-f(x))$[6]	$[\approx -0.28,\infty)$	C^∞	No	No	No
SoftExponential[28]		$f(\alpha,x)=\begin{cases}-\frac{\ln(1-\alpha(x+\alpha))}{\alpha} & \text{for } \alpha<0\\ x & \text{for } \alpha=0\\ \frac{e^{\alpha x}-1}{\alpha}+\alpha & \text{for } \alpha>0\end{cases}$	$f'(\alpha,x)=\begin{cases}\frac{1}{1-\alpha(\alpha+x)} & \text{for } \alpha<0\\ e^{\alpha x} & \text{for } \alpha\geq 0\end{cases}$	$(-\infty,\infty)$	C^∞	Yes	Yes	Yes iff $\alpha=0$
Soft Clipping[29]		$f(\alpha,x)=\frac{1}{\alpha}\log\frac{1+e^{\alpha x}}{1+e^{\alpha(x-1)}}$	$f'(\alpha,x)=\frac{1}{2}\sinh\left(\frac{p}{2}\right)\text{sech}\left(\frac{px}{2}\right)\text{sech}\left(\frac{p}{2}(1-x)\right)$	$(0,1)$	C^∞	Yes	No	No
Sinusoid[30]		$f(x)=\sin(x)$	$f'(x)=\cos(x)$	$[-1,1]$	C^∞	No	No	Yes
Sinc		$f(x)=\begin{cases}1 & \text{for } x=0\\ \frac{\sin(x)}{x} & \text{for } x\neq 0\end{cases}$	$f'(x)=\begin{cases}0 & \text{for } x=0\\ \frac{\cos(x)}{x}-\frac{\sin(x)}{x^2} & \text{for } x\neq 0\end{cases}$	$[\approx -.217234,1]$	C^∞	No	No	No
Gaussian		$f(x)=e^{-x^2}$	$f'(x)=-2xe^{-x^2}$	$(0,1]$	C^∞	No	No	No

^ Here, H is the Heaviside step function.
^ α is a stochastic variable sampled from a uniform distribution at training time and fixed to the expectation value of the distribution at test time.
^ ^ ^ Here, σ is the logistic function.
^ $\alpha > 0$ for the range to hold true.

The following table lists activation functions that are not functions of a single fold x from the previous layer or layers:

Name	Equation	Derivatives	Range	Order of continuity
Softmax	$f_i(\vec{x}) = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}}$ for i = 1, …, J	$\frac{\partial f_i(\vec{x})}{\partial x_j} = f_i(\vec{x})(\delta_{ij} - f_j(\vec{x}))$[7]	$(0,1)$	C^∞
Maxout[31]	$f(\vec{x}) = \max_i x_i$	$\frac{\partial f}{\partial x_j} = \begin{cases} 1 & \text{for } j = \underset{i}{\operatorname{argmax}}\, x_i \\ 0 & \text{for } j \neq \underset{i}{\operatorname{argmax}}\, x_i \end{cases}$	$(-\infty,\infty)$	C^0

^ Here, δ_{ij} is the Kronecker delta.

See also

- Logistic function
- Rectifier (neural networks)
- Stability (learning theory)
- Softmax function

References

1. "What is an Activation Function?" (https://deepai.org/machine-learning-glossary-and-terms/activation-function). *deepai.org*.
2. Hinkelmann, Knut. "Neural Networks, p. 7" (http://didattica.cs.unicam.it/lib/exe/fetch.php?media=didattica:magistrale:kebi:ay_1718:ke-11_neural_networks.pdf) (PDF). *University of Applied Science Northwestern Switzerland*.
3. Hodgkin, A. L.; Huxley, A. F. (1952-08-28). "A quantitative description of membrane current and its application to conduction and excitation in nerve" (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1392413). *The Journal of Physiology*. **117** (4): 500–544. PMC 1392413 (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1392413). PMID 12991237 (https://www.ncbi.nlm.nih.gov/pubmed/12991237).
4. Haykin, Simon S. (1999). *Neural Networks: A Comprehensive Foundation* (https://books.google.com/books?id=bX4pAQAAMAAJ). Prentice Hall. ISBN 978-0-13-273350-2.
5. Cybenko, G.V. (2006). "Approximation by Superpositions of a Sigmoidal function" (https://books.google.com/books?id=4RtVAAAAMAAJ&pg=PA303). In van Schuppen, Jan H. *Mathematics of Control, Signals, and Systems*. Springer International. pp. 303–314.
6. Snyman, Jan (3 March 2005). *Practical Mathematical Optimization: An Introduction to Basic Optimization Theory and Classical and New Gradient-Based Algorithms* (https://books.google.com/books?id=0tFmf_UKl7oC). Springer Science & Business Media. ISBN 978-0-387-24348-1.
7. Wu, Huaiqin (2009). "Global stability analysis of a general class of discontinuous neural networks with linear growth activation functions" (http://linkinghub.elsevier.com/retrieve/pii/S0020025509002539). *Information Sciences*. **179** (19): 3432–3441. doi:10.1016/j.ins.2009.06.006 (https://doi.org/10.1016%2Fj.ins.2009.06.006).
8. Sussillo, David; Abbott, L. F. (2014-12-19). "Random Walk Initialization for Training Very Deep Feedforward Networks". arXiv:1412.6558 (https://arxiv.org/abs/1412.6558) [cs.NE (https://arxiv.org/archive/cs.NE)].
9. Elliot, David L. (1993), "A better activation function for artificial neural networks" (http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.46.7204), *ISR Technical Report TR 93-8, University of Maryland, College Park, MD 20742.*
10. "elliotsig, Elliot symmetric sigmoid transfer function" (https://www.mathworks.com/help/deeplearning/ref/elliotsig.html), *command introduced in Matlab R2012b, Matlab Documentation, MathWorks.*
11. Wuraola, Adedamola; Patel, Nitish (2018), "SQNL:A New Computationally Efficient Activation Function" (https://ieeexplore.ieee.org/document/8489043/), *2018 International Joint Conference on Neural Networks (IJCNN)*, Rio Rio de Janeiro, Brazil: IEEE, pp. 1–7
12. Bergstra, James; Desjardins, Guillaume; Lamblin, Pascal; Bengio, Yoshua (2009). "Quadratic polynomials learn better image features". Technical Report 1337" (http://www.iro.umontreal.ca/~lisa/publications2/index.php/attachments/single/205). *Département d'Informatique et de Recherche Opérationnelle, Université de Montréal.*
13. Glorot, Xavier; Bengio, Yoshua (2010), "Understanding the difficulty of training deep feedforward neural networks" (http://jmlr.org/proceedings/papers/v9/glorot10a/glorot10a.pdf) (PDF), *International Conference on Artificial Intelligence and Statistics (AISTATS'10)*, Society for Artificial Intelligence and Statistics
14. Carlile, Brad; Delamarter, Guy; Kinney, Paul; Marti, Akiko; Whitney, Brian (2017-11-09). "Improving Deep Learning by Inverse Square Root Linear Units (ISRLUs)". arXiv:1710.09967 (https://arxiv.org/abs/1710.09967) [cs.LG (https://arxiv.org/archive/cs.LG)].
15. Nair, Vinod; Hinton, Geoffrey E. (2010), "Rectified Linear Units Improve Restricted Boltzmann Machines" (http://dl.acm.org/citation.cfm?id=3104322.3104425), *27th International Conference on International Conference on Machine Learning*, ICML'10, USA: Omnipress, pp. 807–814, ISBN 9781605589077
16. Eidnes, Lars; Nøkland, Arild (2018). "Shifting Mean Activation Towards Zero with Bipolar Activation Functions" (https://arxiv.org/abs/1709.04054). *International Conference on Learning Representations (ICLR) Workshop*.
17. Maas, Andrew L.; Hannun, Awni Y.; Ng, Andrew Y. (June 2013). "Rectifier nonlinearities improve neural network acoustic models" (https://pdfs.semanticscholar.org/367f/2c63a6f6a10b3b64b8729d601e69337ee3cc.pdf) (PDF). *Proc. ICML*. **30** (1). Retrieved 2 January 2017.
18. He, Kaiming; Zhang, Xiangyu; Ren, Shaoqing; Sun, Jian (2015-02-06). "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". arXiv:1502.01852 (https://arxiv.org/abs/1502.01852) [cs.CV (https://arxiv.org/archive/cs.CV)].
19. Xu, Bing; Wang, Naiyan; Chen, Tianqi; Li, Mu (2015-05-04). "Empirical Evaluation of Rectified Activations in Convolutional Network". arXiv:1505.00853 (https://arxiv.org/abs/1505.00853) [cs.LG (https://arxiv.org/archive/cs.LG)].
20. Clevert, Djork-Arné; Unterthiner, Thomas; Hochreiter, Sepp (2015-11-23). "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)". arXiv:1511.07289 (https://arxiv.org/abs/1511.07289) [cs.LG (https://arxiv.org/archive/cs.LG)].
21. Klambauer, Günter; Unterthiner, Thomas; Mayr, Andreas; Hochreiter, Sepp (2017-06-08). "Self-Normalizing Neural Networks". *Advances in Neural Information Processing Systems*. **30** (2017). arXiv:1706.02515 (https://arxiv.org/abs/1706.02515). Bibcode:2017arXiv170602515K (http://adsabs.harvard.edu/abs/2017arXiv170602515K).
22. Jin, Xiaojie; Xu, Chunyan; Feng, Jiashi; Wei, Yunchao; Xiong, Junjun; Yan, Shuicheng (2015-12-22). "Deep Learning with S-shaped Rectified Linear Activation Units". arXiv:1512.07030 (https://arxiv.org/abs/1512.07030) [cs.CV (https://arxiv.org/archive/cs.CV)].
23. Forest Agostinelli; Matthew Hoffman; Peter Sadowski; Pierre Baldi (21 Dec 2014). "Learning Activation Functions to Improve Deep Neural Networks". arXiv:1412.6830 (https://arxiv.org/abs/1412.6830) [cs.NE (https://arxiv.org/archive/cs.NE)].
24. Glorot, Xavier; Bordes, Antoine; Bengio, Yoshua (2011). "Deep sparse rectifier neural networks" (http://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf) (PDF). *International Conference on Artificial Intelligence and Statistics*.
25. Hendrycks, Dan; Gimpel, Kevin (2016). "Gaussian Error Linear Units (GELUs)". arXiv:1606.08415 (https://arxiv.org/abs/1606.08415) [cs.LG (https://arxiv.org/archive/cs.LG)].
26. Elfwing, Stefan; Uchibe, Eiji; Doya, Kenji (2017). "Sigmoid-Weighted Linear Units for Neural Network Function Approximation in Reinforcement Learning". arXiv:1702.03118 (https://arxiv.org/abs/1702.03118) [cs.LG (https://arxiv.org/archive/cs.LG)].
27. Ramachandran, Prajit; Zoph, Barret; Le, Quoc V (2017). "Searching for Activation Functions". arXiv:1710.05941 (https://arxiv.org/abs/1710.05941) [cs.NE (https://arxiv.org/archive/cs.NE)].
28. Godfrey, Luke B.; Gashler, Michael S. (2016-02-03). "A continuum among logarithmic, linear, and exponential functions, and its potential to improve generalization in neural networks". *7th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management: KDIR*. **1602**: 481–486. arXiv:1602.01321 (https://arxiv.org/abs/1602.01321). Bibcode:2016arXiv160201321G (http://adsabs.harvard.edu/abs/2016arXiv160201321G).
29. Klimek, Matthew D.; Perelstein, Maxim (2018-10-26). "Neural Network-Based Approach to Phase Space Integration". arXiv:1810.11509 (https://arxiv.org/abs/1810.11509) [hep-ph (https://arxiv.org/archive/hep-ph)].
30. Gashler, Michael S.; Ashmore, Stephen C. (2014-05-09). "Training Deep Fourier Neural Networks To Fit Time-Series Data". arXiv:1405.2262 (https://arxiv.org/abs/1405.2262) [cs.NE (https://arxiv.org/archive/cs.NE)].
31. Goodfellow, Ian J.; Warde-Farley, David; Mirza, Mehdi; Courville, Aaron; Bengio, Yoshua (2013). "Maxout Networks". *JMLR Workshop and Conference Proceedings*. **28** (3): 1319–1327. arXiv:1302.4389 (https://arxiv.org/abs/1302.4389). Bibcode:2013arXiv1302.4389G (http://adsabs.harvard.edu/abs/2013arXiv1302.4389G).