

Activation function

From Wikipedia, the free encyclopedia

In computational networks, the **activation function** of a node defines the output of that node given an input or set of inputs. 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, it is the *nonlinear* activation function that allows such networks to compute nontrivial problems using only a small number of nodes. In artificial neural networks this function is also called the transfer function.

Contents

- 1 Functions
 - 1.1 Alternative structures
 - 1.2 Comparison of activation functions
- 2 See also
- 3 References

Functions

In biologically inspired neural networks, the activation function is usually an abstraction representing the rate of action potential firing in the cell. 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 a large number of neurons must be used in computation beyond linear separation of categories.

A line of positive slope may also be used to reflect the increase in firing rate that occurs as input current increases. The function would then 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 also 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 in order 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

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

Activation function having types:

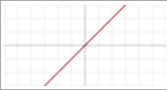


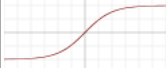
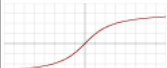
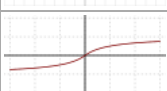

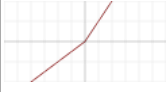

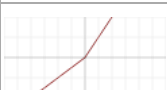



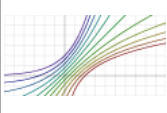
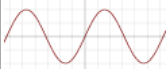
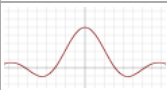

- Identity function.
- Binary step function.
- Bipolar step function.
- Sigmoidal function.
 - Binary sigmoidal function.
 - Bipolar sigmoidal function.
- Ramp function.

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.^[2] 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.
- **Continuously differentiable:** This property is necessary 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.^[3]
- **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.
- **Monotonic:** When the activation function is monotonic, the error surface associated with a single-layer model is guaranteed to be convex.^[4]
- **Smooth** Functions with a **Monotonic derivative** have been shown to generalize better in some cases. The argument for these properties suggests that such activation functions are more consistent with Occam's razor.^[5]
- **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.^[6] 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	Ord conti
Identity		$f(x) = x$	$f'(x) = 1$	$(-\infty, \infty)$	C^∞
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}
Logistic (a.k.a. Soft step)		$f(x) = \frac{1}{1 + e^{-x}}$	$f'(x) = f(x)(1 - f(x))$	$(0, 1)$	C^∞
TanH		$f(x) = \tanh(x) = \frac{2}{1 + e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$	$(-1, 1)$	C^∞
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2 + 1}$	$\left(-\frac{\pi}{2}, \frac{\pi}{2}\right)$	C^∞
Softsign ^{[7][8]}		$f(x) = \frac{x}{1 + x }$	$f'(x) = \frac{1}{(1 + x)^2}$	$(-1, 1)$	C^1
Rectified linear unit (ReLU) ^[9]		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$[0, \infty)$	C^0
Leaky rectified linear unit (Leaky ReLU) ^[10]		$f(x) = \begin{cases} 0.01x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0.01 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$(-\infty, \infty)$	C^0
Parametric rectified linear unit (PReLU) ^[11]		$f(\alpha, x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$(-\infty, \infty)$	C^0
Randomized leaky rectified linear unit (RReLU) ^[12]		$f(\alpha, x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}_{[1]}$	$f'(\alpha, x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$(-\infty, \infty)$	C^0
Exponential linear unit (ELU) ^[13]		$f(\alpha, x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} f(\alpha, x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$(-\alpha, \infty)$	C^1 w $\alpha =$ other C^0
Scaled exponential linear unit (SELU) ^[14]		$f(\alpha, x) = \lambda \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$ with $\lambda = 1.0507$ and $\alpha = 1.67326$	$f'(\alpha, x) = \lambda \begin{cases} f(\alpha, x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$(-\lambda\alpha, \infty)$	C^0
S-shaped rectified linear activation unit (SReLU) ^[15]		$f_{t_l, a_l, t_r, a_r}(x) = \begin{cases} t_l + a_l(x - t_l) & \text{for } x \leq t_l \\ x & \text{for } t_l < x < t_r \\ t_r + a_r(x - t_r) & \text{for } x \geq t_r \end{cases}$ 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
Adaptive piecewise linear (APL) ^[16]		$f(x) = \max(0, x) + \sum_{s=1}^S a_s^* \max(0, -x + b_s^*)$	$f'(x) = H(x) - \sum_{s=1}^S a_s^* H(-x + b_s^*)^{[2]}$	$(-\infty, \infty)$	C^0
SoftPlus ^[17]		$f(x) = \ln(1 + e^x)$	$f'(x) = \frac{1}{1 + e^{-x}}$	$(0, \infty)$	C^∞
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^∞
SoftExponential ^[18]		$f(\alpha, x) = \begin{cases} -\frac{\ln(1-\alpha(x+\alpha))}{\alpha} & \text{for } \alpha < 0 \\ x & \text{for } \alpha = 0 \\ \frac{e^{\alpha x} - 1}{\alpha} + \alpha & \text{for } \alpha > 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} \frac{1}{1-\alpha(x+\alpha)} & \text{for } \alpha < 0 \\ e^{\alpha x} & \text{for } \alpha \geq 0 \end{cases}$	$(-\infty, \infty)$	C^∞
Sinusoid		$f(x) = \sin(x)$	$f'(x) = \cos(x)$	$[-1, 1]$	C^∞
Sinc		$f(x) = \begin{cases} 1 & \text{for } x = 0 \\ \frac{\sin(x)}{x} & \text{for } x \neq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x = 0 \\ \frac{\cos(x)}{x} - \frac{\sin(x)}{x^2} & \text{for } x \neq 0 \end{cases}$	$[\approx -0.217234, 1]$	C^∞
Gaussian		$f(x) = e^{-x^2}$	$f'(x) = -2xe^{-x^2}$	$(0, 1]$	C^∞

^ 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.

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 $i = 1, \dots, J$	$\frac{\partial f_i(\vec{x})}{\partial x_j} = f_i(\vec{x})(\delta_{ij} - f_j(\vec{x}))^{[3]}$	(0, 1)	C^∞
Maxout ^[19]	$f(\vec{x}) = \max_i x_i$	$\frac{\partial f}{\partial x_j} = \begin{cases} 1 & \text{for } j = \operatorname{argmax}_i x_i \\ 0 & \text{for } j \neq \operatorname{argmax}_i x_i \end{cases}$	$(-\infty, \infty)$	C^0

^ Here, *δ* is the Kronecker delta.

See also

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

References

1. Haykin, Simon(1998). *Neural Networks: A Comprehensive Foundation* (2 ed.). Prentice Hall.ISBN 0-13-273350-1.

2. Cybenko, George. "Approximation by superpositions of a sigmoidal function." *Mathematics of control, signals and systems* 2.4 (1989): 303-314.

3. Snyman, Jan. *Practical mathematical optimization: an introduction to basic optimization theory and classical and new gradient-based algorithms*.vol. 97. Springer Science & Business Media, 2005.

4. Wu, Huaqin. "Global stability analysis of a general class of discontinuous neural networks with linear growth activation functions." *Information Sciences* 179.19 (2009): 3432-3441.

5. Gashler, Michael S., and Stephen C. Ashmore. "Training Deep Fourier Neural Networks to Fit Time-Series Data." *Intelligent Computing in Bioinformatics*. Springer International Publishing, 2014. 48-55,arXiv:1405.2262

6. Sussillo, David, and L. F Abbott. "Random walks: Training very deep nonlinear feed-forward networks with smart initialization." *CoRR*arXiv:1412.6558 (2014): 286.

7. Bergstra, James and Desjardins, Guillaume and Lamblin, Pascal and Bengio, Yoshua. "Quadratic polynomials learn better image features". Technical Report 1337, Département d'Informatique et de Recherche Opérationnelle, Université de Montréal 2009, <http://www.iro.umontreal.ca/~lisa/publications2/index.php/attachments/single/205>

8. Glorot, Xavier and Bengio, Yoshua."Understanding the difficulty of training deep feedforward neural networks". In *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10)*. Society for Artificial Intelligence and Statistics. 2010<http://jmlr.org/proceedings/papers/v9/glorot10a/glorot10a.pdf>

9. Nair, Vinod, and Geoffrey E. Hinton. "Rectified linear units improve restricted boltzmann machines." *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*. 2010.

10. 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.

11. arXiv:1502.01852

12. Xu, Bing; Wang, Naiyan; Chen, Tianqi; Li, Mu (2015). "Empirical Evaluation of Rectified Activations in Convolutional Networks"arXiv:1505.00853

13. arXiv:1511.07289v3

14. arXiv:1706.02515v2

15. Jin, Xiaojie; Xu, Chunyan; Feng, Jiashi; Wei, Yunchao; Xiong, Junjun; Yan, Shuicheng (2015). "Deep Learning with S-shaped Rectified Linear Activation Units."arXiv:1512.07030

16. Forest Agostinelli; Matthew Hofman; Peter Sadowski; Pierre Baldi (21 Dec 2014)"Learning Activation Functions to Improve Deep Neural Networks"<https://arxiv.org/abs/1412.6830>. arXiv. Retrieved 2 January 2017.

17. Glorot, Xavier Antoine Bordes, and Yoshua Bengio. "Deep sparse rectifier neural networks" *International Conference on Artificial Intelligence and Statistics*. 2011

18. Luke B. Godfrey and Gashler Michael S. A Continuum among Logarithmic, Linear and Exponential Functions, and Its Potential to Improve Generalization in Neural Networks. In *Proceedings of the 7th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management: KDIR*, pages 481-486. Lisbon, Portugal, November, 2015, arXiv:1602.01321.

19. Ian J. Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, Yoshua Bengio (2013). "Maxout Networks."arXiv:1302.4389

Retrieved from "https://en.wikipedia.org/w/index.php?title=Activation_function&oldid=788625102"

Categories: Artificial neural networks

- This page was last edited on 2 July 2017, at 14:20.
- Text is available under the Creative Commons Attribution-ShareAlike License; additional terms may apply. By using this site, you agree to the Terms of Use and Privacy Policy. Wikipedia® is a registered trademark of the Wikimedia Foundation, Inc., a non-profit organization.