Hàm kích hoạt

Source:

* Introduction to Activation Functions in Neural Networks - Datacamp
* Activation Functions | Fundamentals of Deep Learning – Analytics Vidhya
* Activation Function in Neural Networks - Analytics Vidhya

# I. Hàm kích hoạt (Activation Functions)

## 1. Giới thiệu hàm kích hoạt

Hàm kích hoạt là một thành phần cốt lõi của mạng nơ-ron, giúp chúng học được các mẫu dữ liệu phức tạp. Chúng biến đổi tín hiệu đầu vào của một nút trong mạng nơ-ron thành tín hiệu đầu ra, sau đó được truyền đến lớp tiếp theo.

Nếu không có hàm kích hoạt, mạng nơ-ron sẽ chỉ có thể mô hình hóa các mối quan hệ tuyến tính giữa đầu vào và đầu ra. Hàm kích hoạt đưa vào tính phi tuyến, cho phép mạng nơ-ron học được các ánh xạ phức tạp giữa đầu vào và đầu ra.

Việc lựa chọn hàm kích hoạt phù hợp rất quan trọng để huấn luyện mạng nơ-ron có khả năng tổng quát hóa tốt và đưa ra dự đoán chính xác.

## 2. Cách sử dụng hàm kích hoạt Trong một mạng nơ-ron, mỗi nơ-ron sẽ thực hiện hai phép tính:

1. Tổng tuyến tính của các đầu vào: với đầu vào có trọng số và bias . Tổng tuyến tính được tính bằng:
2. Tính toán hàm kích hoạt: Phép tính này quyết định xem một nơ-ron có nên được kích hoạt hay không bằng cách tính tổng trọng số và cộng thêm bias. Mục đích của hàm kích hoạt là đưa tính phi tuyến vào đầu ra của nơ-ron.

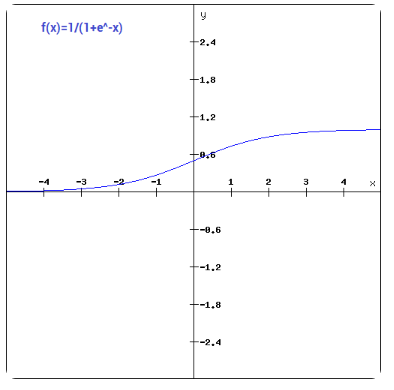
Hầu hết các mạng nơ-ron bắt đầu bằng việc tính tổng trọng số của các đầu vào. Mỗi nút trong một lớp có thể có trọng số riêng biệt. Tuy nhiên, hàm kích hoạt lại giống nhau trên tất cả các nút trong lớp đó. Chúng thường có dạng cố định, trong khi các trọng số được coi là tham số học (learning parameters).

# II.Hàm Kích Hoạt Sigmoid (Sigmoid Activation Function)

## 1. Giới thiệu

Hàm sigmoid, thường được ký hiệu là , là một hàm trơn liên tục và khả vi, đóng vai trò quan trọng trong lịch sử phát triển của mạng nơ-ron. Về mặt toán học, hàm sigmoid được định nghĩa như sau:

Đồ thị hàm sigmoid



Nhận xét

* Hàm này nhận đầu vào là một giá trị thực và cho đầu ra trong khoảng từ 0 đến 1.
* Đồ thị của hàm sigmoid có dạng hình chữ "S", tiệm cận về 0 khi đầu vào là số âm rất lớn và tiệm cận về 1 khi đầu vào là số dương rất lớn.
* Do đầu ra nằm trong khoảng xác suất (0,1), hàm sigmoid thường được sử dụng một cách tự nhiên cho các bài toán phân loại nhị phân (binary classification).

Triển khai trong Python

import numpy as np

def sigmoid\_function(x):

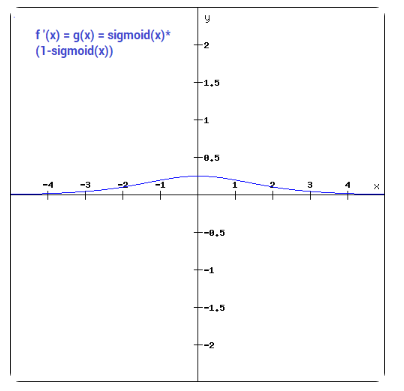
return 1 / (1 + np.exp(-x))

print(sigmoid\_function(7), sigmoid\_function(-22))

# Output: (0.9990889488055994,2.7894680920908113e-10)

Đạo hàm của hàm sigmoid

Đồ thị đạo hàm của hàm sigmoid



Nhận xét

* Các giá trị gradient có độ lớn đáng kể trong khoảng từ đến , nhưng ở các vùng khác đồ thị trở nên phẳng hơn.
* Điều này có nghĩa là những giá trị lớn hơn hoặc nhỏ hơn sẽ có gradient rất nhỏ. Khi giá trị gradient tiến gần đến 0, mạng thực sự không học được gì.

## 2. Ưu điểm

* Đầu ra nằm trong khoảng (0, 1): Hữu ích cho bài toán phân loại nhị phân
* Liên tục và khả vi: Có thể tính gradient để tối ưu bằng lan truyền ngược (backpropagation).

## 3. Nhược điểm

Hàm sigmoid gặp phải vấn đề gọi là vanishing gradient, đặc biệt khi mạng nơ-ron có nhiều lớp (deep neural networks). Khi đầu vào có giá trị quá lớn (dương hoặc âm), hàm sigmoid bão hòa (saturate) tại 0 hoặc 1. Trong những trường hợp này, gradient trở nên cực kỳ nhỏ, dẫn đến:

* Cập nhật trọng số rất chậm trong quá trình lan truyền ngược.
* Các lớp đầu mạng (early layers) hầu như không học được gì, khiến việc huấn luyện trở nên kém hiệu quả hoặc thậm chí dừng lại.

## 4. Ứng dụng chính

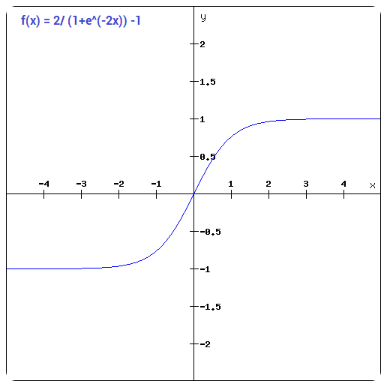
Ngày nay, hàm sigmoid chủ yếu được sử dụng ở lớp đầu ra của các mô hình phân loại nhị phân, giúp chuyển đổi đầu ra thành xác suất thuộc về một lớp cụ thể.

# III. Hàm Kích Hoạt Tanh (Hyperbolic Tangent)

## 1. Giới thiệu

Hàm tanh (tangent hyperbolic) được định nghĩa như sau:

Đồ thị hàm tanh



Nhận xét: Hàm này cho đầu ra trong khoảng −1 đến +1, giúp xử lý tốt hơn các giá trị âm so với hàm sigmoid (vốn chỉ nằm trong khoảng 0 đến 1).

Triển khai trong Python

import numpy as np

def tanh\_function(x):

z = (2/(1 + np.exp(-2\*x))) -1

return z

print(tanh\_function (0.5), tanh\_function (-1))

# Output: (0.4621171572600098, -0.7615941559557646)

Đạo hàm của hàm tanh

Đồ thị đạo hàm của hàm tanh

A graph of a function

AI-generated content may be incorrect.

Nhận xét: Gradient của hàm dốc hơn so với hàm .

## 2. Ưu điểm so với Sigmoid

- Đối xứng qua gốc tọa độ (zero-centered):

* Đầu ra của tanh cân bằng quanh điểm 0. Sự cân bằng này giúp điều chỉnh các cập nhật trọng số hiệu quả hơn trong quá trình tối ưu bằng gradient descent.

- Gradient mạnh hơn:

* Gradient của tanh thường lớn hơn so với sigmoid.
* Điều này giúp mạng nơ-ron học nhanh hơn và giảm thiểu phần nào vấn đề vanishing gradients.

## 3. Nhược điểm

Vẫn gặp vấn đề tiêu biến gradient. Dù tốt hơn sigmoid, hàm tanh vẫn có thể bị bão hòa (saturate) khi đầu vào quá lớn hoặc quá nhỏ, dẫn đến gradient gần 0. Đặc biệt trong các mạng sâu (deep networks), điều này khiến:

* Gradient truyền ngược trở nên cực kỳ nhỏ.
* Các lớp đầu mạng khó cập nhật trọng số, làm chậm quá trình huấn luyện hoặc dẫn đến hội tụ kém.

## 4. Ứng dụng

* Tanh thường được dùng trong các lớp ẩn (hidden layers) của mạng nơ-ron, đặc biệt khi dữ liệu được chuẩn hóa về trung bình 0 (zero-mean).
* Trong hầu hết trường hợp, tanh được ưu tiên hơn sigmoid do tính chất zero-centered và gradient mạnh hơn.
* Tuy nhiên, lựa chọn cuối cùng phụ thuộc vào bài toán cụ thể và kết quả thử nghiệm ban đầu.

# IV. Hàm Kích Hoạt ReLU (Rectified Linear Unit)

## 1. Giới thiệu

Hàm ReLU có dạng đơn giản:

Nó hoạt động bằng cách:

* Trả về 0 nếu đầu vào âm .
* Giữ nguyên giá trị đầu vào nếu dương .

Đồ thị hàm ReLU

#### 

Triển khai Python

def relu\_function(x):

if x<0:

return 0

else:

return x

print(relu\_function (7), relu\_function (-7))

# Output: (7, 0)

Đạo hàm của hàm ReLU

Đồ thị đạo hàm của hàm ReLU

A graphing of a function

AI-generated content may be incorrect.

## 2. Ưu điểm

Khắc phục Vanishing Gradient:

* + Với , ReLU có đạo hàm luôn bằng , giúp gradient truyền ngược không bị suy giảm qua các lớp.
  + Điều này giải quyết vấn đề nghiêm trọng của sigmoid/tanh trong mạng sâu.

Tính phi tuyến (Non-linearity):

* + Dù có dạng tuyến tính khi , ReLU vẫn là hàm phi tuyến do điểm gãy tại .
  + Đặc tính này cho phép mạng nơ-ron học các mẫu phức tạp.

Hiệu quả tính toán:

* + Chỉ cần phép so sánh và gán đơn giản → tốc độ tính toán cực nhanh so với sigmoid/tanh (không cần hàm mũ).
  + Tạo sự kích hoạt thưa (sparse activation): Nhiều nơ-ron trả về 0, giảm đáng kể chi phí tính toán.

## 3. Nhược điểm

Vấn đề Dying ReLU:

* + Nếu một nơ-ron luôn xuất ra 0 (do đầu vào âm liên tục), nó sẽ ngừng học vì gradient tại đó bằng 0.

## 4. Ứng dụng

ReLU là lựa chọn mặc định cho các lớp ẩn (hidden layers) trong hầu hết kiến trúc mạng nơ-ron hiện đại.

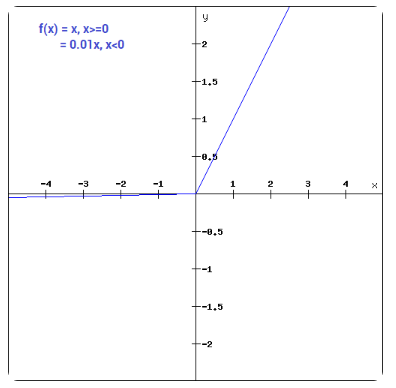
# V. Hàm Kích Hoạt Leaky ReLU

## 1. Giới thiệu

Leaky ReLU là một phiên bản cải tiến của ReLU, khắc phục nhược điểm Dying ReLU bằng cách cho phép một gradient nhỏ khi đầu vào âm .

Công thức toán học

Đồ thị hàm Leaky ReLU



Triển khai Python

def leaky\_relu\_function(x, alpha=0.01):

if x < 0:

return alpha \* x

else:

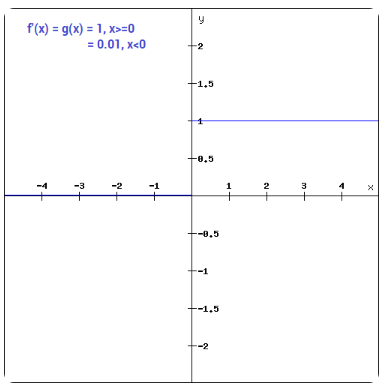
return x

print(leaky\_relu\_function(7), leaky\_relu\_function (-7))

# Output: (7, -0.07)

Đạo hàm của hàm Leaky ReLU

Đồ thị đạo hàm của hàm Leaky ReLU



## 2. Ưu điểm so với ReLU

Khắc phục Dying ReLU:

* Khi đầu vào , , vì thế trọng số vẫn được cập nhật

Giữ được tính kích hoạt thưa (sparse activation):

* Nhiều nơ-ron trả về 0, giảm đáng kể chi phí tính toán.

## 3. Ứng dụng

* Mạng rất sâu (ví dụ: Transformer) nơi Dead ReLU dễ xảy ra.
* Dữ liệu có nhiều giá trị âm

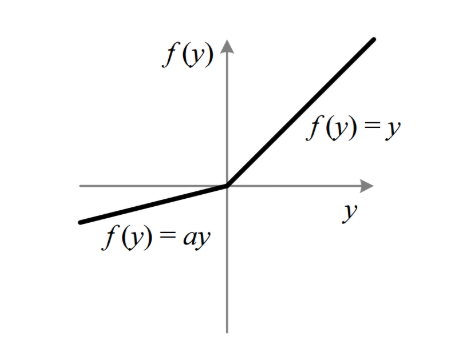
# VI.Hàm Kích Hoạt Parametric ReLU (PReLU)

## 1. Giới thiệu

Parametric ReLU (PReLU) là một phiên bản cải tiến của Leaky ReLU, trong đó hệ số cho phần âm không cố định mà được học tự động trong quá trình huấn luyện. Điều này giúp tối ưu hóa hiệu suất của mạng cho từng bài toán cụ thể.

Công thức toán học

Đồ thị hàm Parametric ReLU



Đạo hàm của hàm Parametric ReLU

## 2. Ưu điểm

Tối ưu hóa tự động: được tự động điều chỉnh phù hợp với dự liệu -> cả thiện độ chính xác

## 3. Nhược điểm

Cần lưu trữ và cập nhật thêm tham số  → Tăng dung lượng mô hình.

## 4. Ứng dụng

Hàm ReLU tham số hóa (Parametric ReLU - PReLU) được sử dụng khi hàm Leaky ReLU vẫn không khắc phục được hiện tượng Dying ReLU

# VII. Hàm Kích Hoạt ELU (Exponential Linear Unit)

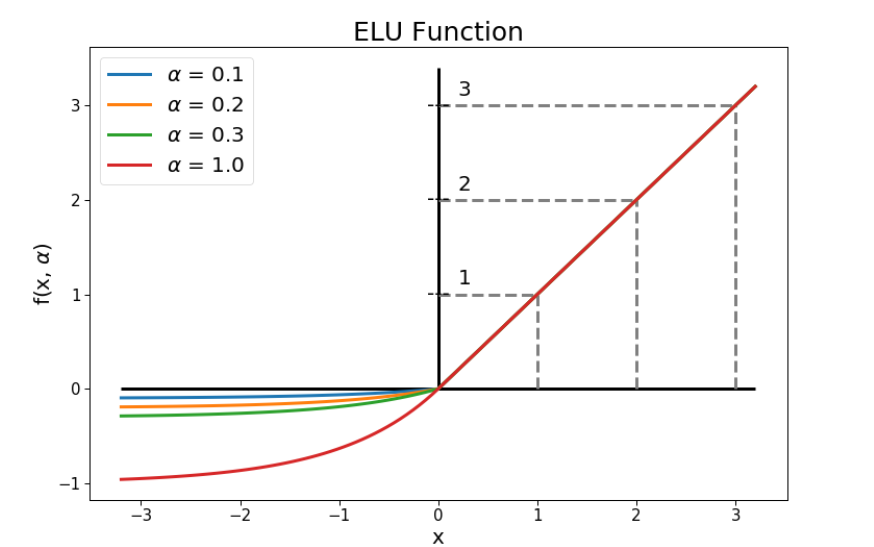
## 1. Giới thiệu

ELU (Exponential Linear Unit) là một biến thể ReLU, khắc phục hiện tượng Dying ReLU bằng cách sử dụng hàm mũ cho giá trị âm.

Công thức toán học

* Với

Đồ thị hàm ELU



Triển khai python

import numpy as np

def elu\_function(x, a):

if x<0:

return a\*(np.exp(x)-1)

else:

return x

print(elu\_function(5, 0.1), elu\_function(-5, 0.1))

# Output: (5, -0.09932620530009145)

Đạo hàm của hàm ELU

## 2. Ưu điểm

Khắc phục Dead ReLU:

* Giá trị âm được xử lý nhờ hàm mũ → Tránh hiện tượng Dying ReLU.

## 3. Nhược điểm

Tính toán phức tạp hơn:

Sử dung hàm -> tốn chi phí hơn so với .

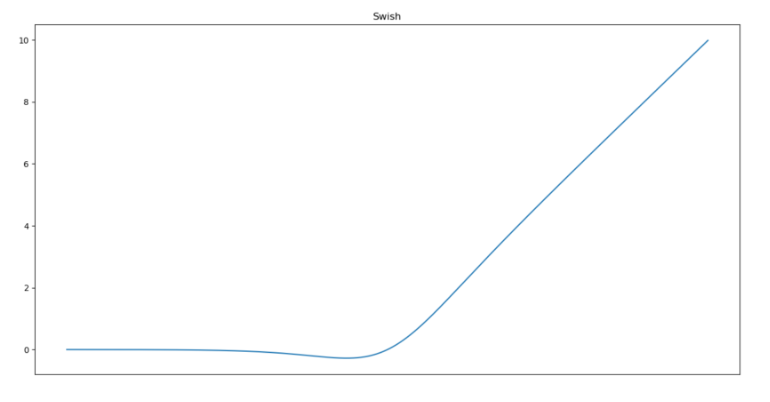
# VIII. Hàm Kích Hoạt Swish

## 1. Giới thiệu

Swish là một hàm kích hoạt ít được biết đến hơn, được phát hiện bởi các nhà nghiên cứu tại Google. Swish thể hiện hiệu suất vượt trội hơn ReLU trên các mô hình sâu hơn. Công thức của Swish:

* là hàm Sigmoid
* Miền giá trị

Đồ thị hàm Swish



Nhận xét

* Đồ thị của hàm này liên tục và hàm có thể vi phân được tại mọi điểm. Đặc tính này rất hữu ích trong quá trình tối ưu hóa mô hình và được coi là một trong những lý do giúp hàm swish vượt trội so với ReLU.
* Một điểm đặc biệt về hàm này là hàm swish không đơn điệu. Điều này có nghĩa là giá trị của hàm có thể giảm ngay cả khi giá trị đầu vào đang tăng.

Triển khai Python

import numpy as np

def swish\_function(x):

return x/(1-np.exp(-x))

print(swish\_function(-67), swish\_function(4))

# Output: (5.35e-28, 4.0746)

## 2.Ưu điểm

Tính chất hàm:

* Swish: hàm liên tục và khả vi tại mọi điểm, không đơn điệu
* ReLU: hàm tuyến tính từng khúc

Xử lý giá trị âm:

* Swish cho phép xử lý giá trị âm
* ReLU ép tất cả giá trị âm về 0

## 3. Nhược điểm

Tốn chi phí tính toán hơn ReLU:

* Cần tính hàm

# IX. Hàm Kích Hoạt Softmax

## 1. Giới thiệu

Hàm softmax (còn gọi là hàm mũ chuẩn hóa) là một công cụ quan trọng cho các bài toán phân loại đa lớp (multi-class classification). Nó biến đổi một vector đầu vào (thường gọi là logits - điểm số dự đoán từ các lớp trước đó của mạng) thành một phân phối xác suất.

* Công thức toán học

Với vector đầu vào , softmax được định nghĩa:

* : Đầu vào của nơ-ron thứ (logit)
* Mẫu số: Tổng hàm mũ của tất cả logits -> Chuẩn hóa thành phân phối xác suất
* Ví dụ minh họa:

Lớp đầu ra có 3 nơ-ron, mỗi nơ-ron tương ứng một lớp.

Công thức Softmax cho nơ-ron thứ

Đầu vào (logits) là:

Tính hàm mũ (exponential) cho từng logit

Tính tổng hàm mũ

Chuẩn hóa thành xác suất

Kết quả

→ Xác suất lần lượt là 42%, 31%, 27% (tổng = 100%).

* Triển khai Python

import numpy as np

def softmax\_function(x):

z = np.exp(x)

z\_ = z/z.sum()

return z\_

print(softmax\_function([0.8, 1.2, 3.1]))

# Output: [0.0802, 0.1197, 0.8001]

## 2. Đặc điểm nổi bật

- Đầu ra là xác suất:

* + Mỗi phần tử đầu ra nằm trong khoảng và tổng cả vector bằng → có thể hiểu là xác suất thuộc về từng lớp.
  + Hàm mũ eˣ đảm bảo đầu ra luôn dương, phù hợp với tính chất xác suất.

- Khuếch đại sự khác biệt:

* + Softmax phóng đại sự chênh lệch giữa các giá trị đầu vào (Ngay cả những khác biệt nhỏ trong giá trị đầu vào cũng có thể dẫn đến sự khác biệt đáng kể trong xác suất đầu ra.).
  + Lớp có điểm số cao nhất sẽ chiếm xác suất áp đảo, giúp mô hình đưa ra quyết định rõ ràng.

- Ứng dụng chính:

* + Thường dùng ở lớp đầu ra của mạng nơ-ron cho bài toán phân loại đa lớp (ví dụ: nhận diện chữ số MNIST).
  + Xác suất đầu ra có thể xem như độ tin cậy của mô hình với từng dự đoán.

- Nhạy cảm với giá trị ngoại lai (outliers):

* + Nếu một giá trị đầu vào quá lớn, softmax sẽ "dồn hết xác suất" vào lớp đó, khiến mô hình trở nên quá tự tin (overconfident).