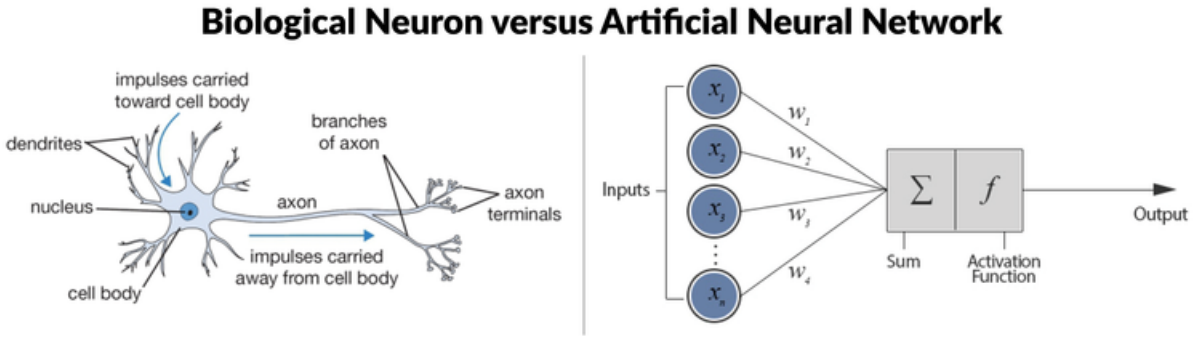
AI VIET NAM – WARMUP 2023

Python Basics – Exercise

Ngày 3 tháng 12 năm 2023

I Giới Thiệu Lý Thuyết và Bài Tập



Hình 1: Mối liên hệ của activation function giữa Biological Neuron và Artificial Neural Network

Giới Thiệu Về Activation Function: Activation functions trong mạng neuron nhân tạo là một khái niệm quan trọng mà người mới bắt đầu học về trí tuệ nhân tạo và học máy cần hiểu. Hãy tưởng tượng một mạng neuron nhân tạo giống như một hệ thống phức tạp mà ở đó thông tin được xử lý và truyền qua nhiều lớp neuron khác nhau. Mỗi neuron trong mạng này nhận đầu vào, thực hiện tính toán nhất định, và sau đó tạo ra một đầu ra.

Activation function chính là "trái tim" của mỗi neuron, quyết định liệu một neuron có nên được kích hoạt hay không. Có nhiều loại activation functions khác nhau, mỗi loại có những đặc tính và ứng dụng riêng. Ví dụ, một số function tạo ra đầu ra dựa trên ngưỡng nhất định (ví dụ: nếu đầu vào vượt quá một giá trị nhất định, neuron sẽ được kích hoạt), trong khi những function khác có thể xử lý thông tin một cách mềm dẻo hơn.

Tóm lại, activation functions đóng vai trò quan trọng trong việc quyết định cách thông tin được xử lý và truyền đi trong mạng neuron nhân tạo, giúp mô hình học máy có khả năng học được những mô hình phức tạp từ dữ liệu.

Phần bài tập này sẽ hướng dẫn các bạn làm quen với 8 loại activation function cơ bản và phổ biến trong mạng neuron, thông qua việc đọc và hiểu code Python.

1. **Sigmoid Function:** Sigmoid Function, hay còn được gọi là logistic function, là một trong những activation functions cơ bản nhất trong machine learning và neural networks. Hình dạng của nó giống như chữ "S" nằm ngang. Sigmoid chuyển đổi mọi giá trị đầu vào thành một giá trị đầu ra nằm giữa 0 và 1, với một sự chuyển đổi mượt mà tại giá trị 0.

   Ứng Dụng: Sigmoid Function thường được sử dụng trong các bài toán phân loại nhị phân, nơi mà mục tiêu là phân loại đầu vào thành một trong hai lớp.

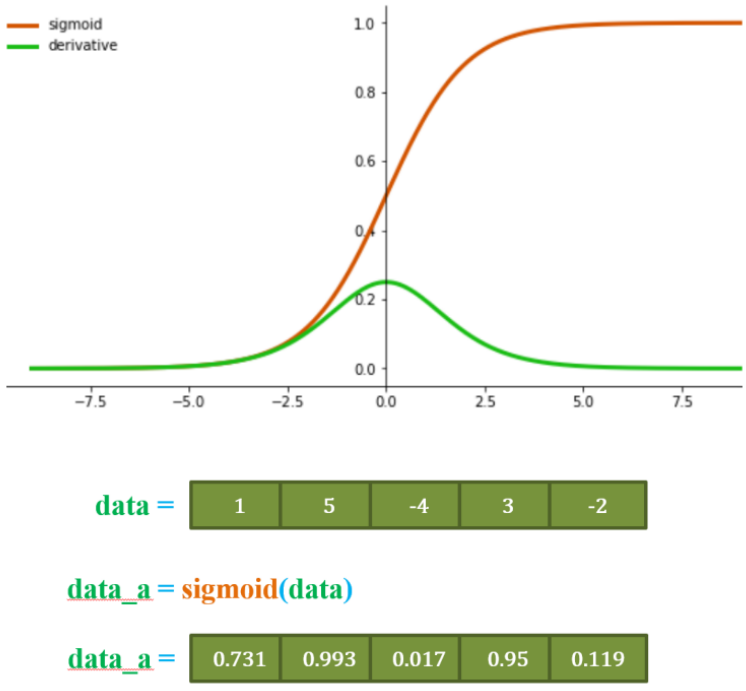
   Ưu Điểm:

   - **Dễ hiểu và triển khai:** Do tính chất đơn giản và phổ biến, Sigmoid rất được triển khai trong nhiều loại mạng neuron.
   - **Đầu ra nằm trong khoảng [0,1]:** Giá trị đầu ra luôn nằm trong khoảng từ 0 đến 1, giúp dễ dàng diễn giải như là xác suất.

   Nhược điểm:

   - **Vanishing gradient problem**: Khi đầu vào có giá trị lớn hoặc nhỏ, đạo hàm của Sigmoid tiệm cận đến 0, dẫn đến vấn đề vanishing gradient, làm chậm quá trình học của mạng.
   - **Tâm đối xứng không tại 0**: Tâm đối xứng không nằm tại điểm 0, điều này có thể gây ra vấn đề trong việc điều chỉnh trọng số trong quá trình học.

$$sigmoid(x) = \frac{1}{1 + e^{-x}} \tag{1}$$



Hình 2: Hàm Sigmoid và đạo hàm

Yêu cầu: Dựa vào công thức và ví dụ về Sigmoid, hãy thử viết lại Sigmoid function sử dụng Python.

2. **Tanh Function:** Tanh Function, viết tắt từ hyperbolic tangent function, là một hàm toán học phổ biến trong machine learning và neural networks. Tanh giúp chuyển đổi giá trị đầu vào thành khoảng giá trị nằm trong [-1, 1].

   Ứng Dụng: Tanh Function thường được sử dụng như một hàm kích hoạt trong neural networks và có nhiều ứng dụng quan trọng. Ví dụ dược áp dụng trong các lớp ẩn của neural networks để giúp mô hình học các biểu diễn phức tạp.

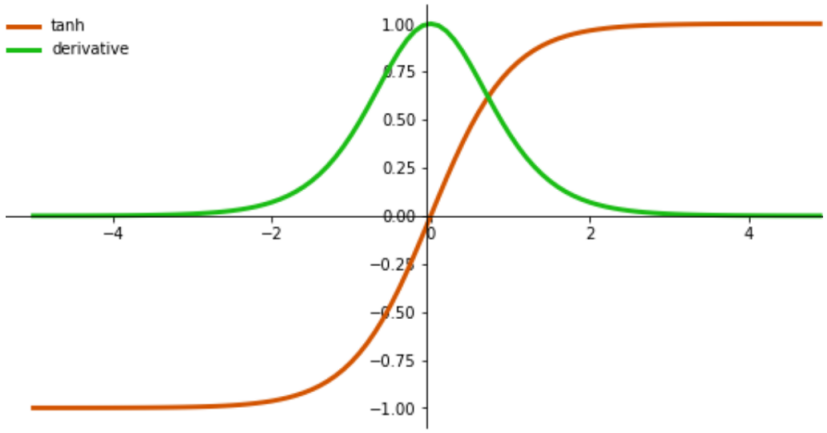
   Ưu Điểm:

   - **Zero-Centered Output:** Giá trị đầu ra của Tanh nằm gần trung bình 0, giúp ổn định quá trình học.
   - **Giảm vanishing gradient:** Hỗ trợ giảm nguy cơ vanishing gradient, tăng cường khả năng học của mô hình.

   Nhược điểm:

   - **Bão Hòa (Saturation):** Có khả năng bị saturation ở giá trị cực đại hoặc cực tiểu.
   - **Không Zero-Centered cho Dữ Liệu Đầu Vào Không Được Chuẩn Hóa:** Nếu dữ liệu đầu vào không được chuẩn hóa, giá trị trung bình của đầu ra có thể không gần 0.

$$tanh(x) = \frac{2}{1 + e^{-2x}} - 1 \tag{2}$$



data = | 1 | 5 | -4 | 3 | -2 |

data_a = tanh(data)

data_a = | 0.761 | 0.999 | -0.999 | 0.995 | -0.964 |

Hình 3: Hàm Tanh và đạo hàm

Yêu cầu: Dựa vào công thức và ví dụ về Tanh, hãy thử viết lại Tanh function sử dụng Python.

3. **Softplus Function:** Softplus Function là một hàm toán học thường được sử dụng trong machine learning và neural networks. Softplus giúp chuyển đổi giá trị đầu vào thành một khoảng không giới hạn trong đoạn $(0, +\infty)$.

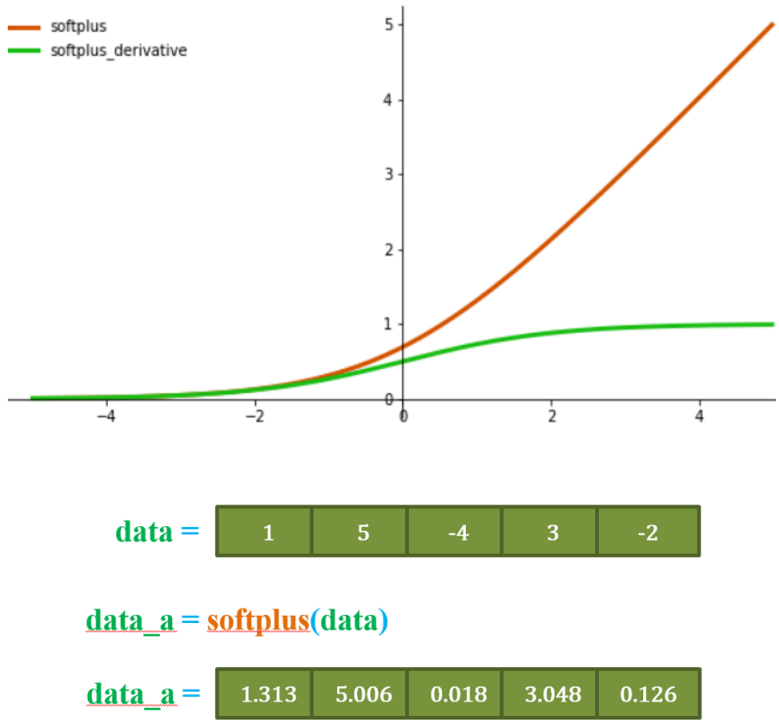
   Ứng Dụng: Softplus Function thường được sử dụng trong các tình huống đòi hỏi một hàm kích hoạt có dạng tăng nhanh và không giữ giá trị đầu ra trong một khoảng cụ thể. Softplus thường được sử dụng làm hàm kích hoạt trong các lớp ẩn của neural networks, đặc biệt là trong các mô hình mà yêu cầu tính không giới hạn của đầu ra.

   Ưu Điểm:

   - **Không giới hạn đầu Ra:** Cho phép giá trị đầu ra không giới hạn và tăng lên nhanh chóng với giá trị đầu vào dương.
   - **Khả năng xấp xỉ hàm ReLU:** Trong một số trường hợp, Softplus có khả năng xấp xỉ hàm Rectified Linear Unit (ReLU).

   Nhược điểm: Có khả năng bị bão hòa (saturation) và dễ mất đi ở giá trị âm đối với giá trị đầu vào lớn.

$$Softplus(x) = log(1 + e^x) \tag{3}$$



Hình 4: Hàm Softplus và đạo hàm

Yêu cầu: Dựa vào công thức và ví dụ về Softplus, hãy thử viết lại Softplus function sử dụng Python.

4. **ReLU Function:** ReLU, viết tắt của "Rectified Linear Unit", là một hàm kích hoạt rất phổ biến trong neural networks. Được đánh giá cao vì tính đơn giản nhưng hiệu quả, ReLU được sử dụng rộng rãi để giúp neural networks học được những đặc trưng phức tạp mà không gặp phải vấn đề biến mất gradient. ReLU hoạt động dựa trên nguyên tắc rất đơn giản: nếu đầu vào là số âm, hàm sẽ trả về giá trị 0, còn nếu đầu vào là số dương, hàm sẽ trả về chính giá trị đó.

   Ứng dụng: ReLU phổ biến trong các ứng dụng như nhận dạng hình ảnh và xử lý ngôn ngữ tự nhiên, nơi ReLU giúp cải thiện tốc độ học và giảm thiểu vấn đề biến mất gradient. ReLU cũng rất quan trọng trong học tăng cường và các tác vụ phân loại, cung cấp một phương pháp hiệu quả để xử lý thông tin phi tuyến tính.

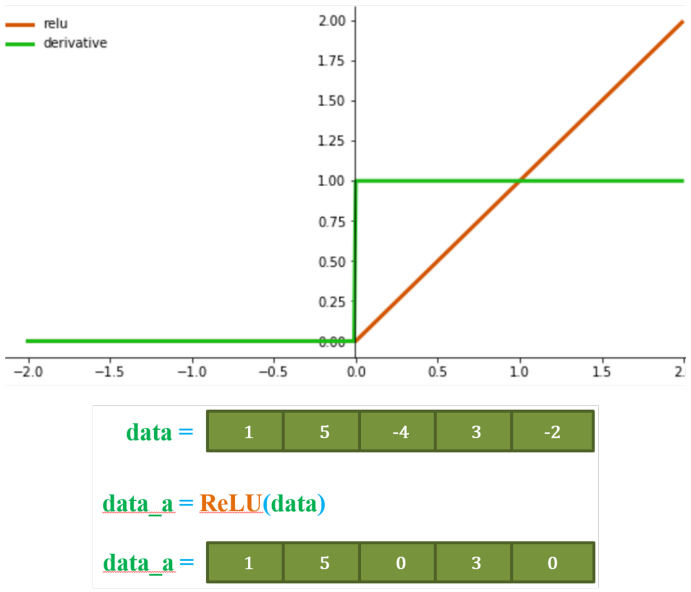
   Ưu điểm:

   - **Tính toán đơn giản**: Do cấu trúc đơn giản, ReLU nhanh và hiệu quả hơn trong việc tính toán so với các hàm kích hoạt phi tuyến tính khác.
   - **Giảm mất mát gradient**: ReLU giúp giảm thiểu vấn đề biến mất gradient, một điểm mạnh quan trọng trong quá trình huấn luyện neural networks.

   Nhược điểm:

   - **Vấn đề dying ReLU**: Đôi khi neuron có thể chỉ trả về giá trị 0 cho tất cả các đầu vào, dẫn đến hiện tượng "dying ReLU", khi đó neuron trở nên không hoạt động.
   - **Không đối xứng tại zero**: Do ReLU không phải là hàm có giá trị trung bình bằng 0 nên có thể gây ra vấn đề trong quá trình tối ưu hóa mạng.

$$relu(x) = \begin{cases} 0 & if\ x \leq 0 \\ x & if\ x > 0. \end{cases} \tag{4}$$



Hình 5: Hàm ReLU và đạo hàm

Yêu cầu: Dựa vào công thức và ví dụ về ReLU, hãy thử viết lại ReLU function sử dụng Python.

5. **LeakyReLU Function:** LeakyReLU, một biến thể của hàm ReLU, cũng là một hàm kích hoạt phi tuyến tính được sử dụng trong neural networks. Điểm đặc biệt của LeakyReLU là không trả về giá trị 0 cho các giá trị đầu vào âm, mà thay vào đó là một giá trị nhỏ, thường được đặt là $0.01x$.

   Ứng dụng: LeakyReLU được sử dụng rộng rãi trong xử lý ảnh và nhận dạng hình ảnh, cung cấp khả năng học các đặc trưng phức tạp mà không gặp phải vấn đề "dying ReLU". Trong lĩnh vực học tăng cường và xử lý ngôn ngữ tự nhiên, LeakyReLU giúp cải thiện đáng kể hiệu suất của các mô hình bằng cách giảm thiểu biến mất gradient, một vấn đề thường gặp trong quá trình huấn luyện mạng nơ-ron sâu.

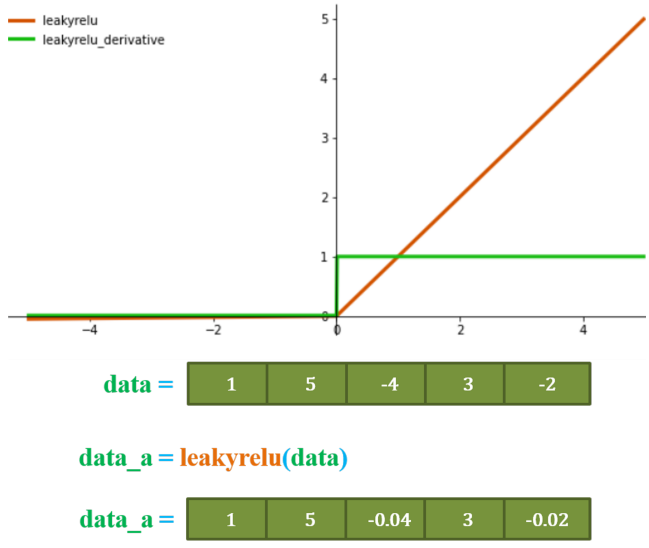
   Ưu điểm:

   - **Giảm thiểu vấn đề dying ReLU**: Bằng cách cho phép trả về một giá trị nhỏ thay vì 0 cho các đầu vào âm, LeakyReLU giúp giảm thiểu vấn đề các neuron trở nên không hoạt động.
   - **Phù hợp cho deep neural network**: Tương tự như ReLU, LeakyReLU cũng giúp giải quyết vấn đề biến mất gradient, làm cho việc huấn luyện deep neural network trở nên hiệu quả hơn.

   Nhược điểm:

   - **Dùng giá trị cứng 0.01**: Việc dùng giá trị 0.01 làm cho activation này không có tính tổng quát.
   - **Không đối xứng tại zero**: Tương tự như ReLU, LeakyReLU cũng không phải là hàm có giá trị trung bình bằng 0, có thể gây ra vấn đề trong quá trình tối ưu hóa mạng.

$$LeakyRelu(x) = \begin{cases} 0.01x & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases} \tag{5}$$



Hình 6: Hàm LeakyReLU và đạo hàm

Yêu cầu: Dựa vào công thức và ví dụ về LeakyReLU, hãy thử viết lại LeakyReLU function sử dụng Python.

6. **ELU Function:** ELU, viết tắt của Exponential Linear Unit, là một loại activation function được sử dụng trong neural network. Được đề xuất bởi Djork-Arné Clevert và các cộng sự vào năm 2015, ELU nhằm mục đích giải quyết một số vấn đề của các activation functions trước đây như ReLU. ELU có công thức (6) giúp ELU giảm thiểu vấn đề vanishing gradient ở các giá trị âm, đồng thời vẫn duy trì tính phi tuyến cần thiết cho quá trình học sâu.

   Ứng Dụng: ELU chủ yếu được sử dụng trong các mạng neuron sâu, đặc biệt là những nơi cần giải quyết vấn đề vanishing gradient. ELU thường được áp dụng trong các mô hình học sâu phức tạp như convolutional neural networks (CNNs) và recurrent neural networks (RNNs) để cải thiện tốc độ học và hiệu suất của mô hình.

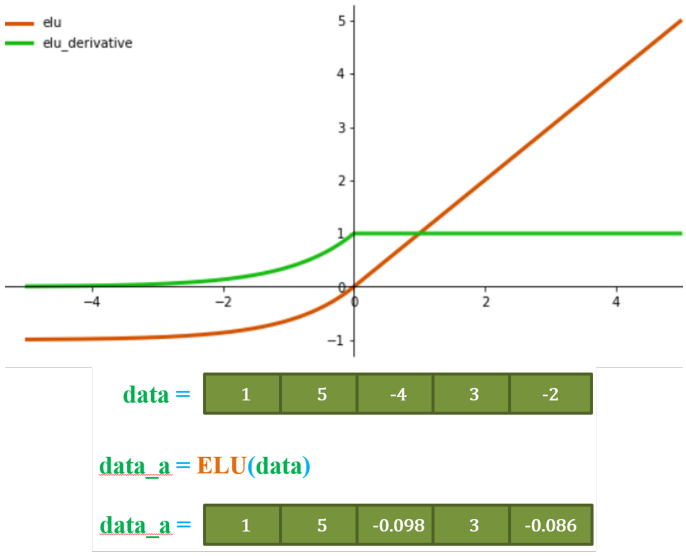
   Ưu Điểm:

   - **Hiệu suất cao:** Trong một số trường hợp, ELU cho thấy hiệu suất tốt hơn so với các activation functions khác như ReLU và Leaky ReLU, đặc biệt trong các mạng sâu.
   - **Có đầu ra âm:** Việc này giúp duy trì một phân phối đầu ra cân bằng hơn, có thể cải thiện khả năng học của mô hình.

   Nhược Điểm:

   - **Tính toán phức tạp hơn:** Do cấu trúc phức tạp của công thức, ELU đòi hỏi nhiều chi phí tính toán hơn so với ReLU.
   - **Lựa chọn α:** Việc lựa chọn giá trị của α có thể ảnh hưởng đáng kể đến hiệu suất của mô hình và nó không có một quy tắc cụ thể nào, đòi hỏi phải thử nghiệm để tìm ra giá trị phù hợp.

$$ELU(x) = \begin{cases} \alpha(e^x - 1) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases} \tag{6}$$



Hình 7: Hàm ELU và đạo hàm

Yêu cầu: Dựa vào công thức và ví dụ về ELU, hãy thử viết lại ELU function sử dụng Python ($\alpha = 0.1$).

7. **PReLU Function:** PReLU, viết tắt của Parametric Rectified Linear Unit, là một biến thể của ReLU trong neural network. Được giới thiệu trong một nghiên cứu năm 2015, PReLU làm cho hệ số độ dốc của phần âm của ReLU có thể học được trong quá trình huấn luyện, thay vì là một hằng số cố định như trong Leaky ReLU. Ở đây, α là một tham số học được, không phải là một giá trị cố định. Điều này giúp PReLU linh hoạt hơn so với ReLU và Leaky ReLU

   Ứng Dụng: PReLU thường được sử dụng trong các mô hình học sâu, đặc biệt là trong các mạng neuron sâu như CNNs và RNNs. PReLU giúp cải thiện hiệu suất của mô hình trong các tác vụ phức tạp như nhận dạng hình ảnh, xử lý ngôn ngữ tự nhiên, và học tăng cường.

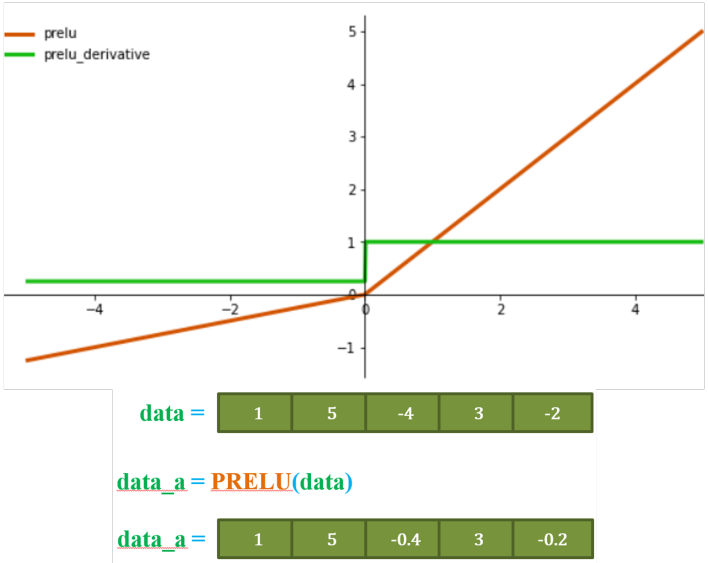
   Ưu Điểm:

   - **Tính linh hoạt cao:** Do α là tham số học được, PReLU có khả năng tự điều chỉnh dựa trên dữ liệu, giúp cải thiện hiệu suất.
   - **Hiệu suất tốt trong mô hình sâu:** Trong nhiều trường hợp, PReLU đã chứng minh có hiệu suất tốt hơn ReLU, đặc biệt trong các mô hình sâu và phức tạp.

   Nhược Điểm:

   - **Nguy cơ overfitting:** Do tính chất tham số hóa, nếu không được điều chỉnh cẩn thận, PReLU có thể dẫn đến overfitting, đặc biệt khi sử dụng trong các mạng có ít dữ liệu.
   - **Tăng chi phí tính toán:** Việc học các tham số α thêm vào chi phí tính toán và thời gian huấn luyện của mô hình.
   - **Cần tinh chỉnh thêm:** Việc tìm ra giá trị tối ưu của α đòi hỏi quá trình thử nghiệm và tinh chỉnh, có thể làm phức tạp quá trình huấn luyện.

$$PReLU(x) = \begin{cases} \alpha x & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases} \tag{7}$$



Hình 8: Hàm PReLU và đạo hàm

Yêu cầu: Dựa vào công thức và ví dụ về PReLU, hãy thử viết lại PReLU function sử dụng Python. (α=0.1)

8. **Swish Function:** Swish Function là một loại activation function được phát triển bởi các nhà nghiên cứu tại Google. Được giới thiệu lần đầu trong một bài báo năm 2017, Swish đã nhanh chóng thu hút sự chú ý trong cộng đồng học máy vì tính hiệu quả và độc đáo của nó. Công thức của Swish (8), trong đó x là đầu vào, và β là một tham số có thể học nhưng được dùng với mặc định là 1.0. Swish kết hợp lợi ích của việc sử dụng thông tin đầu vào (như ReLU) và tránh được dying ReLU.

   Ứng Dụng: Swish thường được sử dụng trong các mô hình học sâu, như CNNs và RNNs. Swish đặc biệt hữu ích trong các tác vụ liên quan đến học sâu và học tăng cường, nơi cần một activation function hiệu quả và linh hoạt.

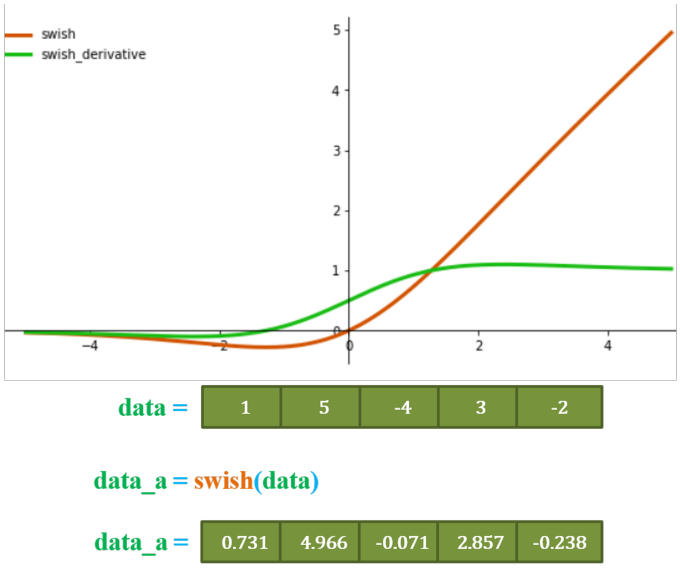
   Ưu Điểm:

   - **Hiệu suất cao trong mô hình sâu:** Swish thường cho thấy hiệu suất tốt trong các mạng neuron sâu, đôi khi vượt trội so với các activation function truyền thống như ReLU.
   - **Liên Tục và Khả Vi:** Điều này giúp trong quá trình cập nhật trọng số qua backpropagation diễn ra mượt mà hơn so với các hàm như ReLU.

   Nhược Điểm:

   - **Tính toán phức tạp hơn:** So với các activation functions đơn giản như ReLU, Swish có công thức phức tạp hơn, dẫn đến tăng chi phí tính toán.
   - **Ít Thông Dụng:** Do là hàm mới hơn so với các hàm kích hoạt truyền thống như ReLU và Sigmoid, Swish chưa được sử dụng rộng rãi và nghiên cứu kỹ lưỡng.

$$Swish(x) = \frac{x}{1 + e^{-\beta x}} = x sigmoid(\beta x) \tag{8}$$



Hình 9: Hàm Swish và đạo hàm

Yêu cầu: Dựa vào công thức và ví dụ về Swish, hãy thử viết lại Swish function sử dụng Python. (β=1)

II. Câu Hỏi Trắc Nghiệm

1. Đoạn code dưới đây đang thực hiện activation function nào?

```
beta = 1.0
x = -4
y = x * (1 / (1 + math.e**(-beta*x)))
print(y)
>> -0.07194483984836625
```

(A). ReLU
(B). Sigmoid
(C). PReLU
(D). Swish

2. Đoạn code dưới đây đang thực hiện activation function nào?

```
alpha = 0.1
x = -4
if x<=0:
    y = x*alpha*(x<=0)
else:
    y = x
print(y)
>> -0.4
```

(A). Swish
(B). Sigmoid
(C). PReLU
(D). ReLU

3. Đoạn code dưới đây đang thực hiện activation function nào?

```
import math

x = 2
y = math.log(1 + math.exp(x))
print(y)
>> 2.1269280110429727
```

(A). Softplus
(B). Sigmoid
(C). PReLU
(D). Swish

4. Đoạn code dưới đây đang thực hiện activation function nào?

```
import math

x = 2
y = 2/(1 + math.exp(-2*x)) - 1
print(y)
>> 0.9640275800758169
```

(A). ReLU
(B). Sigmoid
(C). PReLU
(D). Tanh

5. Đoạn code dưới đây đang thực hiện activation function nào?

```
import math

x = 2
y = 1 / (1 + math.exp(-x))
print(y)
>> 0.8807970779778823
```

(A). ReLU
(B). Sigmoid
(C). PReLU
(D). Tanh

6. Đoạn code dưới đây đang thực hiện activation function nào?

```
x = -2.0
if x<=0:
    y = 0.0
else:
    y = x
print(y)
>> 0.0
```

(A). Swish
(B). LeakyReLU
(C). ReLU
(D). Tanh

7. Đoạn code dưới đây đang thực hiện activation function nào?

```
x = -1.2
if x<=0:
    y = x*0.01
else:
    y = x
print(y)
>> -0.012
```

(A). Sigmoid
(B). LeakyReLU
(C). ReLU
(D). ELU

8. Đoạn code dưới đây đang thực hiện activation function nào?

```
import math

alpha = 0.1
x = -4
if x<=0:
    y = alpha*(math.e**x-1)
else:
    y = x
>> -0.09816843611112658
```

(A). Sigmoid
(B). Tanh
(C). ReLU
(D). ELU