AI VIET NAM – AI COURSE 2025

Exercise: Activation Functions

Quoc-Thai Nguyen, Quang-Vinh Dinh

I. Giới thiệu

1. Giới thiệu về Python

Python là một ngôn ngữ lập trình cấp cao, thông dịch, đa mục đích và có cú pháp rõ ràng, dễ học. Nó hỗ trợ nhiều mô hình lập trình như hướng thủ tục, hướng đối tượng và hàm. Python được sử dụng rộng rãi trong khoa học dữ liệu, trí tuệ nhân tạo, phát triển web, tự động hóa và nhiều lĩnh vực khác.

2. Biến và Kiểu Dữ Liệu

Biến là vùng nhớ được đặt tên dùng để lưu trữ giá trị. Python là ngôn ngữ động nên không cần khai báo kiểu trước.

Một số kiểu dữ liệu cơ bản:

- `int`: số nguyên (*e.g.* 1, -5)
- `float`: số thực (*e.g.* 3.14, -0.5)
- `str`: chuỗi ký tự (*e.g.* "hello")
- `bool`: giá trị đúng/sai (True/False)
- `list, tuple, set, dict`: cấu trúc dữ liệu phức tạp

Ví dụ khai báo:

```
x = 10
name = "An"
flag = True
```

3. Câu lệnh điều kiện và vòng lặp

if, elif, else Cho phép kiểm tra điều kiện để điều khiển luồng chương trình:

```
if x > 0:
    print("Số dương")
elif x == 0:
    print("Số không")
else:
    print("Số âm")
```

for Vòng lặp duyệt qua dãy số hoặc phần tử:

```
for i in range(5):
    print(i)
```

while Lặp lại khi điều kiện còn đúng:

```
x = 0
while x < 5:
    print(x)
    x += 1
```

4. Hàm trong Python

Hàm là khối mã thực thi khi được gọi. Có thể truyền tham số và nhận giá trị trả về.

Khai báo và sử dụng hàm:

```
def greet(name):
    return "Hello " + name

print(greet("An"))
```

Tham số mặc định và không giới hạn:

```
def say_hello(name="bạn"):
    print("Chào", name)

def sum_all(*args):
    return sum(args)
```

5. Một số hàm dựng sẵn

- `type(x)`: kiểm tra kiểu
- `len(s)`: độ dài chuỗi/danh sách
- `int()`, `str()`, `float()`: chuyển đổi kiểu
- `range(start, stop)`: tạo cách dang sách số từ start đến stop-1
- `import math`, `import random`

II. Câu hỏi tự luận

1. **Viết hàm tính độ đo F1-Score thường được sử dụng để đánh giá mô hình phân loại.**

   - Precision = $\dfrac{TP}{TP + FP}$
   - Recall = $\dfrac{TP}{TP + FN}$
   - F1-score = $2 * \dfrac{Precision * Recall}{Precision + Recall}$
   - Input: hàm nhận 3 giá trị **tp, fp, fn**
   - Output: trả về và in ra kết quả của **Precision, Recall, và F1-score**

   Chú ý: Đề bài yêu cầu các điều kiện sau

   - Phải **kiểm tra giá trị nhận vào tp, fp, fn là kiểu dữ liệu int**, nếu là kiểu dữ liệu khác khác thì in ra thông báo cho người dùng ví dụ kiểm tra fn là float, in ra thông báo **"fn must be int"** và thoát hàm hoặc dừng chương trình.
   - Yêu cầu **tp, fp, fn phải đều lớn hơn hoặc bằng 0**, nếu khác thì in **"tp and fp and fn must be greater than or equal zero"** và thoát hàm hoặc dừng chương trình. Cần kiểm tra trường hợp phép tính chi 0.

```
# Examples
evaluate_f1_components(tp=2, fp=3, fn=4)
>>> precision is  0.4
recall is  0.3333333333333333
f1-score is  0.3636363636363636

evaluate_f1_components(tp="a", fp=3, fn=4)
>>> tp must be int


evaluate_f1_components(tp=2, fp="a", fn=4)
>>> fp must be int

evaluate_f1_components(tp=2, fp=3, fn="a")
>>> tp must be int

evaluate_f1_components(tp=2.1, fp=3, fn=0)
>>> tp must be int
```

2. Viết hàm tính giá trị cho các hàm số kích hoạt.

- **Sigmoid Function:** Sigmoid Function, hay còn được gọi là logistic function, là một trong những hàm kích hoạt cơ bản nhất trong machine learning và neural networks. Hình dạng của nó giống như chữ "S". Sigmoid chuyển đổi mọi giá trị đầu vào thành một giá trị đầu ra nằm trong khoảng 0 và 1.

  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 được triển khai trong nhiều loại mạng neuron.

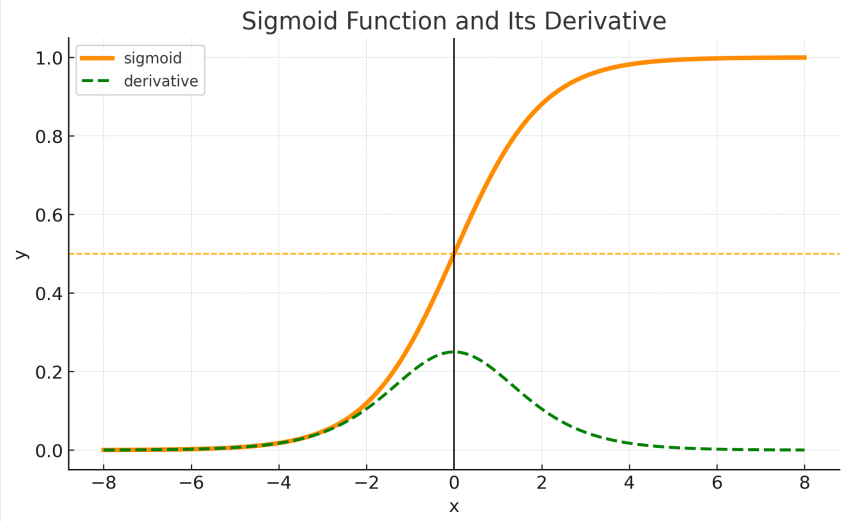
  – **Đầu ra nằm trong khoảng [0,1]:** Giá trị đầu ra luôn nằm trong khoảng từ 0 đến 1, giúp dễ dàng diễn giải như là xác suất.

  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.

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



Hình 1: Hàm Sigmoid và đạo hàm

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.

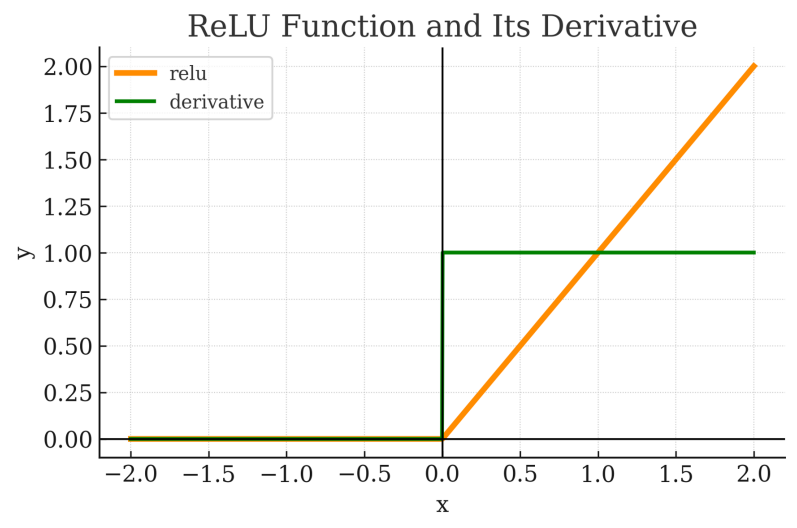
Ưu điểm:

- **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.

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



Hình 2: Hàm ReLU và đạo hàm

- **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 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.

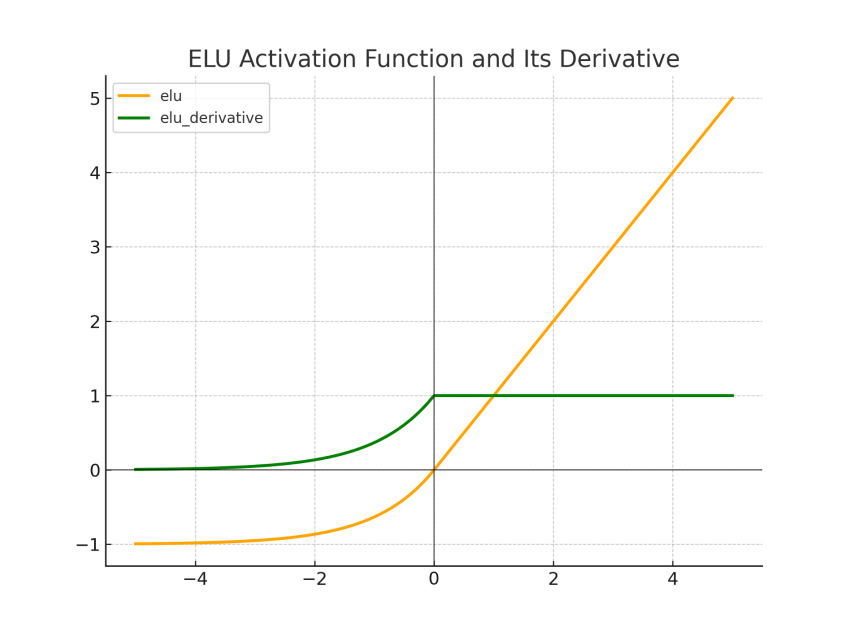
Ưu Điểm:

- **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.

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



Hình 3: Hàm ELU và đạo hàm

- Input:
  - Người dùng nhập giá trị **x**
  - Người dùng nhập tên **activation function chỉ có 3 loại (sigmoid, relu, elu)**

- Output: Kết quả **f(x)** (x khi đi qua activation function tương ứng theo activation function name). Ví dụ **nhập x=3, activation_function = "relu". Output: trả về kết quả và in ra "relu: f(3)=3"**

Chú ý: Lưu ý các điều kiện sau:

- Dùng hàm **is_number** được cung cấp sẵn để **kiểm tra x có hợp lệ hay không** (vd: x=10, is_number(x) sẽ trả về True ngược lại là False). Nếu **không hợp lệ in ra "x must be a number"và dừng chương trình.**
- Kiểm tra **tên hàm kích hoạt có hợp lệ hay không nằm trong 3 tên (sigmoid, relu, elu). Nếu không print "ten_function_user is not supported"** (vd người dùng nhập "belu"thì in ra "belu is not supported")
- Ép kiểu dữ liệu của **x** sang **float**
- Thực hiện theo công thức với activation name tương ứng. In ra kết quả
- Dùng math.e để lấy số e
- $\alpha = 0.01$

```python
# Given
def is_number(n):
    try:
        float(n)   # Type-casting the string to ‘float‘.
                   # If string is not a valid ‘float‘,
                   # it’ll raise ‘ValueError‘ exception
    except ValueError:
        return False
    return True
```

```
interactive_activation_function()
>> Input x = 1.5
Input activation Function (sigmoid|relu|elu): sigmoid
sigmoid: f(1.5) = 0.8175744761936437

interactive_activation_function()
>> Input x = abc
x must be a number

interactive_activation_function()
>> Input x = 1.5
Input activation Function (sigmoid|relu|elu): belu
belu is not supportted
```

3. Viết hàm lựa chọn hàm mất mát và tính giá trị hàm mất mát (loss functions)

- MAE = $\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$
- MSE = $\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$
- RMSE = $\sqrt{MSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$
- **n** chính là **số lượng samples (num_samples)**, với **i** là mỗi sample cụ thể. Ở đây các bạn có thể hiểu là cứ mỗi **i** thì sẽ **có 1 cặp** y_i **là target và** $\hat{y}$ **là predict**.
- Input:
  - Người dùng **nhập số lượng sample (num_samples) được tạo ra (chỉ nhận integer numbers)**
  - Người dùng **nhập loss name (MAE, MSE, RMSE-(optional)) chỉ cần MAE và MSE, bạn nào muốn làm thêm RMSE đều được**.
- Output: Print ra **loss name, sample, predict, target, loss**
  - **loss name:** là loss mà người dùng chọn
  - **sample:** là thứ tự sample được tạo ra (ví dụ num_samples=5, thì sẽ có 5 samples và mỗi sample là sample-0, sample-1, sample-2, sample-3, sample-4)
  - **predict:** là số mà model dự đoán (chỉ cần dùng random tạo random một số trong range [0,10))
  - **target:** là số target mà momg muốn mode dự đoán đúng (chỉ cần dùng random tạo random một số trong range [0,10))
  - **loss:** là kết quả khi đưa predict và target vào hàm loss
  - **note:** ví dụ num_sample=5 thì sẽ có 5 cặp predict và target.

Chú ý: Các bạn lưu ý

- Dùng **.isnumeric() method** để kiểm tra **num_samples** có hợp lệ hay không (vd: x=10, num_samples.isnumeric() sẽ trả về True ngược lại là False). Nếu **không hợp lệ print "number of samples must be an integer number"** và dừng chương trình.
- **Dùng vòng lặp for, lặp lại num_samples lần. Mỗi lần dùng random modules tạo một con số ngẫu nhiên trong range [0.0, 10.0) cho predict và target**. Sau đó đưa predict và target vào loss function và print ra kết quả mỗi lần lặp.
- **Dùng random.uniform(0,10)** để tạo ra một số ngẫu nhiên trong range [0,10)
- **Giả xử người dùng luôn nhập đúng loss name MSE, MAE, và RMSE (đơn giản bước này để các bạn không cần kiểm tra tên hợp lệ)**
- Dùng abs() để tính trị tuyệt đối ví dụ abs(-3) sẽ trả về 3
- Dùng math.sqrt() để tính căn bậc 2

```
cal_activation_function()
>> Input number of samples (integer number) which are generated: 3
Input loss name: MAE
loss_name: MAE, sample: 0: pred: 5.92 target: 6.96 loss: 1.05
loss_name: MAE, sample: 1: pred: 0.1 target: 5.74 loss: 5.64
loss_name: MAE, sample: 2: pred: 0.94 target: 2.1 loss: 1.16
final MAE: 2.6156580640656397

cal_activation_function()
>> Input number of samples (integer number) which are generated: aa
number of samples must be an integer number
```

4. Viết 4 hàm để ước lượng các hàm số sau.

$$\sin(x) \approx \sum_{n=0}^{\infty}(-1)^n \frac{x^{(2n+1)}}{(2n+1)!} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \cdots$$

$$\cos(x) \approx \sum_{n=0}^{\infty}(-1)^n \frac{x^{2n}}{(2n)!} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \frac{x^{10}}{10!} + \cdots$$

$$\sinh(x) \approx \sum_{n=0}^{\infty} \frac{x^{(2n+1)}}{(2n+1)!} = x + \frac{x^3}{3!} + \frac{x^5}{5!} + \frac{x^7}{7!} + \frac{x^9}{9!} + \cdots$$

$$\cosh(x) \approx \sum_{n=0}^{\infty} \frac{x^{2n}}{(2n)!} = 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \frac{x^6}{6!} + \frac{x^8}{8!} + \frac{x^{10}}{10!} + \cdots$$

- Input: x (số muốn tính toán) và n (số lần lặp muốn xấp xỉ)
- Output: Kết quả ước lượng hàm tương ứng với x. Ví dụ hàm cos(x=0) thì output = 1

Chú ý: Các bạn chú ý các điều kiện sau

- x là radian
- n là số nguyên dương > 0
- các bạn nên viết một hàm tính giai thừa riêng

```
approx_sin(x=3.14, n=10)
>> 0.0015926529267151343

approx_cos(x=3.14, n=10)
>> -0.9999987316527259

approx_sinh(x=3.14, n=10)
>> 11.53029203039954

approx_cosh(x=3.14, n=10)
>> 11.573574828234543
```

III. Câu hỏi trắc nghiệm

Câu hỏi 1 : Viết hàm thực hiện đánh giá mô hình phân loại bằng F1-Score. Hàm nhận vào 3 giá trị **tp, fp, fn** và trả về F1-score

- Precision $= \dfrac{TP}{TP + FP}$
- Recall $= \dfrac{TP}{TP + FN}$
- F1-score $= 2 * \dfrac{Precision * Recall}{Precision + Recall}$

Đầu ra của chương trình sau đây là gì?

```python
import math
def compute_f1_score(tp, fp, fn):
    """
    Tính điểm F1 Score từ các giá trị:
    - tp: Số lượng true positives
    - fp: Số lượng false positives
    - fn: Số lượng false negatives

    Công thức:
        precision = tp / (tp + fp)
        recall = tp / (tp + fn)
        f1_score = 2 * (precision * recall) / (precision + recall)

    Trả về:
        f1_score: Điểm F1 Score dưới dạng float
    """
    # Your code here

    # End your code

assert round(calc_f1_score(tp=2, fp=3, fn=5), 2) == 0.33
print(round(calc_f1_score(tp=2, fp=4, fn=5), 2))
```

a) 0.33
b) 0.35
c) 0.31
d) Raise an Error

Câu hỏi 2 : Viết hàm **is_number** nhận input có thể là string hoặc một số **kiểm tra n (một số) có hợp lệ hay không** (vd: n=10, is_number(n) sẽ trả về True ngược lại là False). Đầu ra của chương trình sau đây là gì?

```python
import math
def is_number(n):
    # Your code here

```

```
    # End your code
assert is_number(3) == 1.0
assert is_number("-2a") == 0.0
print(is_number(1))
print(is_number("n"))
```

a) n
b)
True
False
c) 1
d) Raise an Error

Câu hỏi 3 : 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) Elu
b) Sigmoid
c) ReLU
d) Activation function khác

Câu hỏi 4 : Viết function thực hiện Sigmoid Function $f(x) = \frac{1}{1+e^{-x}}$. Nhận input là x và return kết quả tương ứng trong Sigmoid Function. Đầu ra của chương trình sau đây là gì?

```
import math
def calc_sig(x):
    """
    Tính hàm sigmoid: (x) = 1 / (1 + e^(-x))
    """
    # Your code here

    # End your code

assert round(calc_sig(3), 2)==0.95
print(round(calc_sig(2), 2))
```

a) 0.88
b) 2
c) 0.9907970779778823
d) Raise an Error

Câu hỏi 5 : Viết function thực hiện Elu Function $f(x) = \begin{cases} \alpha(e^x - 1) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$. Nhận input là x và return kết quả tương ứng trong Elu Function. Đầu ra của chương trình sau đây là gì khi α = 0.01?

```python
import math
def calc_elu(x):
    """
    Tính hàm ELU (Exponential Linear Unit):
    ELU(x) = x                  nếu x >= 0
           = alpha * (e^x - 1) nếu x < 0
    """
    # Your code here

    # End your code

assert round(calc_elu(1))==1
print(round(calc_elu(-1), 2))
```

a) -1
b) -0.01
c) -0.1063212055882855 76
d) Raise an Error

Câu hỏi 6 : Viết function nhận 2 giá trị x, và tên của activation function act_name **activation function chỉ có 3 loại (sigmoid, relu, elu)**, thực hiện tính toán activation function tương ứng với name nhận được trên giá trị của x và trả kết quả. Đầu ra của chương trình sau đây là gì?

```python
import math

def calc_activation_func(x, act_name):
    """
    Tính hàm kích hoạt cho x dựa trên act_name:
    "relu", "sigmoid", hoặc "elu".
    """
    # Your code here

    # End your code
assert calc_activation_func(x = 1, act_name="relu") == 1
print(round(calc_activation_func(x = 3, act_name="sigmoid"), 2))
```

a) 0.95
b) 4
c) 1
d) Raise an Error

Câu hỏi 7 : Viết function tính absolute error $= |y - \hat{y}|$. Nhận input là y và $\hat{y}$, return về kết quả absolute error tương ứng. Đầu ra của chương trình sau đây là gì?

```python
def calc_ae(y, y_hat):
    """
    Tính sai số tuyệt đối (Absolute Error)
    giữa giá trị thực tế và giá trị dự đoán.

    Tham số:
    y (float hoặc int): Giá trị thực tế (ground truth).
    y_hat (float hoặc int): Giá trị dự đoán.
```

```
    Trả về:
    float: Giá trị tuyệt đối của hiệu giữa y và y_hat.
    """
    # Your code here

    # End your code

y = 1
y_hat = 6
assert calc_ae(y, y_hat)==5
y = 2
y_hat = 9
print(calc_ae(y, y_hat))
```

a) 7
b) 8
c) 9
d) Raise an Error

Câu hỏi 8 : Viết function tính squared error $= (y - \hat{y})^2$. Nhận input là y và $\hat{y}$, return về kết quả squared error tương ứng. Đầu ra của chương trình sau đây là gì?

```
def calc_se(y, y_hat):
    """
    Tính sai số bình phương (Squared Error)
    giữa giá trị thực tế và giá trị dự đoán.

    Tham số:
    y (float hoặc int): Giá trị thực tế (ground truth).
    y_hat (float hoặc int): Giá trị dự đoán.

    Trả về:
    float: Bình phương của hiệu giữa y và y_hat.
    """
  # Your code here

  # End your code
y = 4
y_hat = 2
assert calc_se(y, y_hat) == 4
print(calc_se(2, 1))
```

a) 1
b) 2
c) 3
d) Raise an Error

Câu hỏi 9 : Dựa vào công thức xấp xỉ sin và điều kiện được giới thiệu trong phần tự luận. Viết function xấp xỉ sin khi nhận x là giá trị muốn tính sin(x) và n là số lần lặp muốn xấp xỉ. Return về kết quả sin(x) với bậc xấp xỉ tương ứng. Đầu ra của chương trình sau đây là gì?

```
def approx_sin(x, n):
    """
```

```
    Approximate the sine of x using the Taylor series expansion.

    Parameters:
    x (float): The input angle in radians.
    n (int): Number of terms in the Taylor series expansion.

    Returns:
    float: Approximate value of sin(x) using n+1 terms.
    """
    # Your code here

    # End your code

assert round(approx_sin(x=1, n=10), 4)==0.8415
print(round(approx_sin(x=3.14, n=10), 4))
```

a) 0.0016
b) 0.0017
c) 0.0018
d) Raise an Error

Câu hỏi 10 : Dựa vào công thức xấp xỉ cos và điều kiện được giới thiệu trong phần tự luận. Viết function xấp xỉ cos khi nhận x là giá trị muốn tính cos(x) và n là số lần lặp muốn xấp xỉ. Trả về kết quả cos(x) với bậc xấp xỉ tương ứng. Đầu ra của chương trình sau đây là gì?

```
def approx_cos(x, n):
    """
    Approximate the cosine of x using the Taylor series expansion.
    Parameters:
    x (float): The input angle in radians.
    n (int): Number of terms in the Taylor series expansion.
    Returns:
    float: Approximate value of cos(x) using n+1 terms.
    """
    # Your code here

    # End your code

assert round(approx_cos(x=1, n=10), 2)==0.54
print(round(approx_cos(x=3.14, n=10), 2))
```

a) 2
b) 1
c) -1.0
d) Raise an Error

Câu hỏi 11 : Dựa vào công thức xấp xỉ sinh và điều kiện được giới thiệu trong phần tự luận. Viết function xấp xỉ sinh khi nhận x là giá trị muốn tính sinh(x) và n là số lần lặp muốn xấp xỉ. Trả về kết quả sinh(x) với bậc xấp xỉ tương ứng. Đầu ra của chương trình sau đây là gì?

```
def approx_sinh(x, n):
    """
    Approximate the sinh of x using the Taylor series expansion.
```

```
    Parameters:
    x (float): The input angle in radians.
    n (int): Number of terms in the Taylor series expansion.
    Returns:
    float: Approximate value of sinh(x) using n+1 terms.
    """
    # Your code here

    # End your code

assert round(approx_sinh(x=1, n=10), 2)==1.18
print(round(approx_sinh(x=3.14, n=10), 2))
```

a) 11.53
b) 12.53
c) 13.53
d) Raise an Error

Câu hỏi 12 : Dựa vào công thức xấp xỉ cosh và điều kiện được giới thiệu trong phần tự luận. Viết function xấp xỉ cosh khi nhận x là giá trị muốn tính cosh(x) và n là số lần lặp muốn xấp xỉ. Trả về kết quả cosh(x) với bậc xấp xỉ tương ứng. Đầu ra của chương trình sau đây là gì?

```
def approx_cosh(x, n):
    """
    Approximate the hyperbolic cosine of x using the Taylor series.
    Parameters:
    x (float): The input value.
    n (int): Number of terms in the Taylor series expansion.
    Returns:
    float: Approximate value of cosh(x) using n+1 terms.
    """
    # Your code here

    # End your code
assert round(approx_cosh(x=1, n=10), 2)==1.54
print(round(approx_cosh(x=3.14, n=10), 2))
```

a) 11.57
b) 11.58
c) 11.59
d) Raise an Error

IV. Phụ lục

1. **Hint**: Dựa vào file tải về Exercise: Activation Functions để hoàn thiện các đoạn code.
2. **Solution**: Các file code cài đặt hoàn chỉnh và phần trả lời nội dung trắc nghiệm có thể được tải về tại đây (Lưu ý: Sáng khi hết deadline phần project, admin mới copy các nội dung bài giải nêu trên vào đường dẫn).
3. **Rubric**:

Basic Python – Exercise Rubric		
Câu	**Kiến Thức**	**Đánh Giá**
1	- Hiểu được cách sử dụng Python để tính toán F1-Score và giới thiệu cơ bản đánh giá mô hình bằng F1-Score.	- Có khả năng viết function và xử lý lỗi khi code Python tính F1-Score, thể hiện sự hiểu biết về thực hiện code theo một công thức cho trước.
2	- Nắm được cách thức cài đặt các hàm kích hoạt như sigmoid, relu và elu trong Python.	- Thể hiện khả năng thực hiện các hàm này trong môi trường lập trình Python, phù hợp cho các ứng dụng AI.
3	- Hiểu rõ cách thực hiện các hàm regression loss phổ biến trong Python để tính toán loss cho các bài toán regression.	- Thể hiện khả năng cài đặt các hàm loss đa dạng trong môi trường lập trình Python, phù hợp để tính toán loss trong các bài toán regression
4	- Tìm hiểu về các phương pháp lập trình Python để ước lượng giá trị hàm số.	- Có khả năng lập trình Python để mô phỏng các hàm số, áp dụng hiệu quả trong các bài toán tính toán.
5	- Hiểu được cách xây dựng và sử dụng các chỉ số đánh giá mô hình nâng cao trong Python.	Thực hiện thành thạo các tính toán error phức tạp trong việc sử dụng Python để cải thiện hiệu suất mô hình.

- Hết -