AI VIET NAM – AIO2024

Basic Python – Exercise

Ngày 1 tháng 6 năm 2024

I. Câu hỏi tự luận

Note: Các bạn có thể dùng hàm print để in các variable như sau:

```
# Examples
var1 = 'str_var1'
var2 = 3
var3 = 15.5
print('Print single variable (var1): ', var1)
print(f'Print more than one variables: var1={var1}, var2={var2}, var3={var3}')

>> Print single variable (var1):  str_var1
Print more than one variables: var1=str_var1, var2=3, var3=15.5
```

1. **Viết function thực hiện đánh giá classification model bằng F1-Score.**

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

   NOTE: Đề bài yêu cầu các điều kiện sau

   - Phải **kiểm tra giá trị nhận vào tp, fp, fn là type int**, nếu là type khác thì print ví dụ check fn là float, print **'fn must be int'** và thoát hàm hoặc dừng chương trình.
   - Yêu cầu **tp, fp, fn phải đều lớn hơn 0**, nếu không thì print **'tp and fp and fn must be greater than zero'** và thoát hàm hoặc dừng chương trình

```
# Examples
exercise1(tp=2, fp=3, fn=4)
>> precision is  0.4
recall is  0.3333333333333333
f1-score is  0.3636363636363636

exercise1(tp='a', fp=3, fn=4)
>> tp must be int

```

```
exercise1(tp=2, fp='a', fn=4)
>> fp must be int


exercise1(tp=2, fp=3, fn='a')
>> tp must be int

exercise1(tp=2, fp=3, fn=0)
>> tp and fp and fn must be greater than zero

exercise1(tp=2.1, fp=3, fn=0)
>> tp must be int
```

Code Listing 1: Đây là các ví dụ các bạn không cần thiết đặt tên giống ví dụ

Để mở rộng kiến thức, các bạn có thể tìm hiểu thêm về F1-score, bao gồm ưu điểm của F1-score so với accuracy.

2. **Viết function mô phỏng theo 3 activation function.**

- **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". 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:

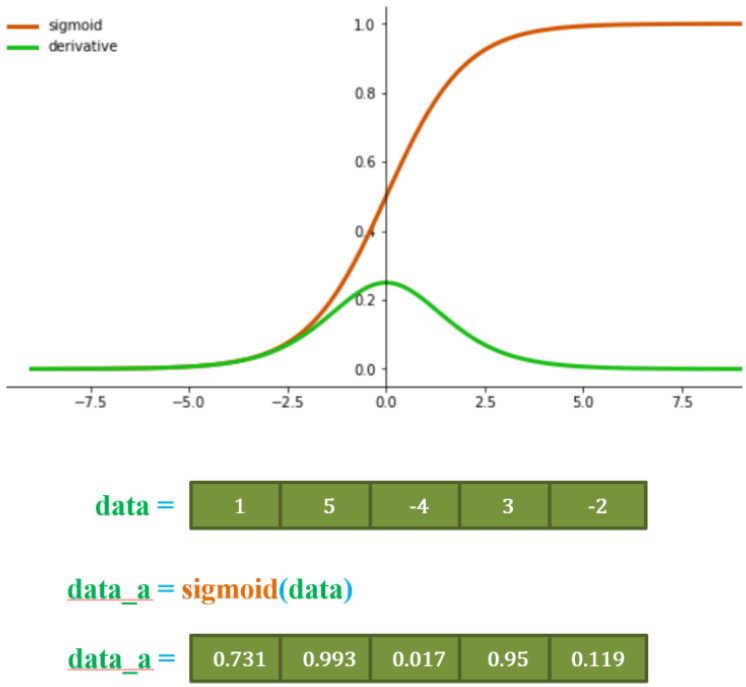
  – **Vanishing gradient problem**: Khi đầu vào có giá trị lớn hoặc nhỏ, đạo hàm của Sigmoid tiệm cận đến 0, dẫn đến vấn đề vanishing gradient, làm chậm quá trình học của mạng.

  – **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.

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

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



Hình 1: Hàm Sigmoid và đạo hàm

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.

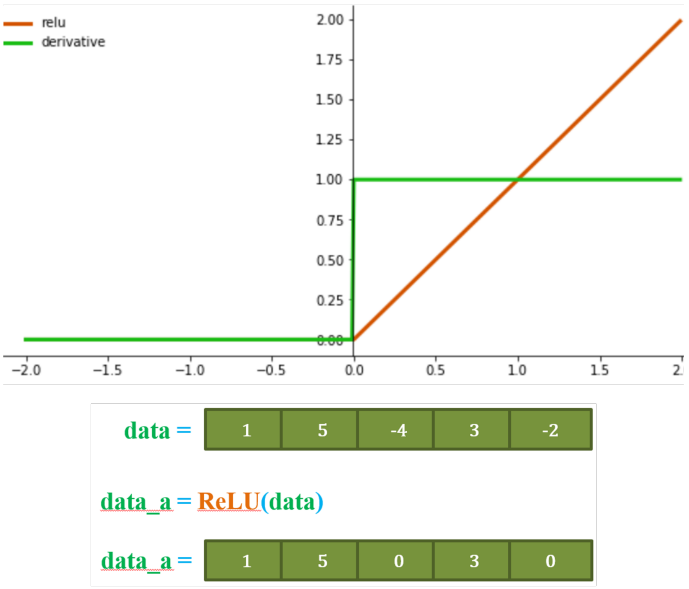
Nhược điểm:

- **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.

$$relu(x) = \begin{cases} 0 & if\ x \leq 0 \\ x & if\ x > 0. \end{cases} \tag{2}$$

- **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



Hình 2: Hàm ReLU và đạo hàm

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:

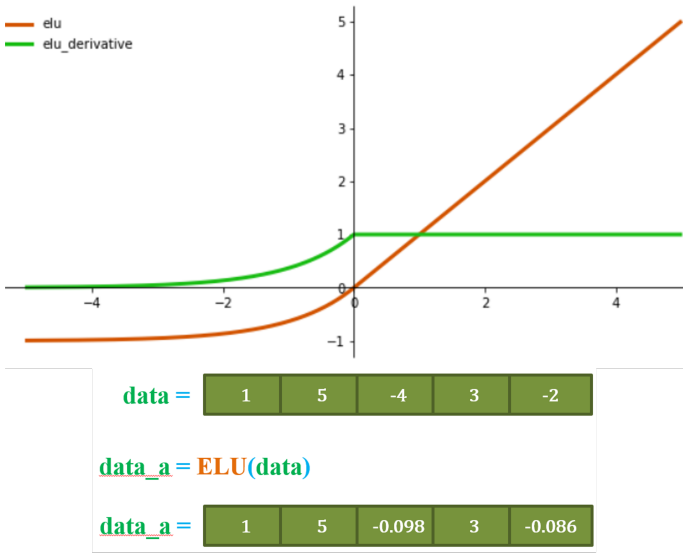
- **Tính toán phức tạp hơn:** Do cấu trúc phức tạp của công thức, ELU đòi hỏi nhiều chi phí tính toán hơn so với ReLU.
- **Lựa chọn α:** Việc lựa chọn giá trị của α có thể ảnh hưởng đáng kể đến hiệu suất của mô hình và nó không có một quy tắc cụ thể nào, đòi hỏi phải thử nghiệm để tìm ra giá trị phù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

$$ELU(x) = \begin{cases} \alpha(e^x - 1) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases} \tag{3}$$

- 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: print 'relu: f(3)=3'**

NOTE: Lưu ý các điều kiện sau:

- Dùng function **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 print 'x must be a number' và dừng chương trình.**



Hình 3: Hàm ELU và đạo hàm

- Kiểm tra **activation function name 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 print 'belu is not supported')
- Convert **x** sang **float** type
- Thực hiện theo công thức với activation name tương ứng. Print 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
```

Code Listing 2: Cho trước hàm is_number

```
exercise2()
>> Input x = 1.5
Input activation Function (sigmoid|relu|elu): sigmoid
sigmoid: f(1.5) = 0.8175744761936437

exercise2()
>> Input x = abc
x must be a number

exercise2()
>> Input x = 1.5
Input activation Function (sigmoid|relu|elu): belu
belu is not supportted
```

Code Listing 3: Dây là các ví dụ các bạn không cần thiết đặt tên giống ví dụ

3. **Viết function lựa chọn regression loss function để tính loss:**

   - 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.

   Note: 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 giảng bước này để các bạn không cần check 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

```
exercise3()
>> Input number of samples (integer number) which are generated: 5
Input loss name: RMSE
loss name: RMSE, sample: 0, pred: 6.659262156575629, target: 4.5905830130732355,
    loss: 4.279433398761796
loss name: RMSE, sample: 1, pred: 4.592264312227207, target: 8.447168720237958,
    loss: 14.860287994900718
loss name: RMSE, sample: 2, pred: 8.701801828625959, target: 9.280646891626386,
    loss: 0.3350616069599687
```

```
loss name: RMSE, sample: 3, pred: 4.799972972282257, target: 9.877147335937869,
    loss: 25.777699518961764
loss name: RMSE, sample: 4, pred: 0.20159822778697878, target: 5.540221923628147,
     loss: 28.50090296579681
final RMSE: 3.8406610234536727


exercise3()
>> Input number of samples (integer number) which are generated: aa
number of samples must be an integer number
```

Code Listing 4: Đây là các ví dụ các bạn không cần thiết đặt tên giống ví dụ

4. Viết 4 functions 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

NOTE: 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
```

Code Listing 5: Đây là các ví dụ các bạn không cần thiết đặt tên giống ví dụ

5. **Viết function thực hiện Mean Difference of n^{th} Root Error:** là một kỹ thuật thông dụng trong các ứng dụng như phát hiện và theo dõi đối tượng. Ngoài ra, phương pháp này cũng có thể được áp dụng cho các bài toán hồi quy khác. Cụ thể, chúng ta sẽ tính căn bậc n của cả y_i và $\hat{y}_i$ trước khi tính toán hàm lỗi, theo công thức sau:

$$\text{MD\_nRE} = \frac{1}{m}\sum_{i=1}^{m}\left(\sqrt[n]{y_i} - \sqrt[n]{\hat{y}_i}\right)^p,$$

trong đó, căn bậc n được áp dụng cho cả y_i và $\hat{y}_i$ trước khi tính loss, và p đại diện cho bậc của hàm loss. Sử dụng hàm MD_nRE với $n = 2$ và $p = 1$ (tương tự như hàm MAE) cho ví dụ trên sẽ cho kết quả như sau: :

y	$\hat{y}$	MAE	MD_nRE ($n = 2, p = 1$)
100	99.5	0.5	0.025
50	49.5	0.5	0.035
20	19.5	0.5	0.056
5.5	5.0	0.5	0.110
1.0	0.5	0.5	0.293
0.6	0.1	0.5	0.458

Bảng kết quả so sánh *Mean Absolute Error* (MAE) và *Mean Difference of nth Root Error* có thể đưa ra một số kết luận:

- **Consistency of MAE:** MAE giữ nguyên ở mức 0.5 qua tất cả các cặp y_i và $\hat{y}_i$. Điều này thể hiện tính chất của MAE trong việc cung cấp một metric đồng nhất về độ lớn của loss mà không phụ thuộc vào tỷ lệ hoặc phân phối của các giá trị.
- **Behavior of MD_nRE:** MD_nRE cho thấy các giá trị thay đổi, giảm dần khi độ lớn của y_i (và $\hat{y}_i$) tăng lên. Điều này chỉ ra rằng MD_nRE, không giống như MAE, nhạy cảm với tỷ lệ của dữ liệu. Đối với các giá trị nhỏ hơn của y_i và $\hat{y}_i$, việc chuyển đổi căn bậc có ảnh hưởng đáng kể hơn đến việc tính toán loss, dẫn đến giá trị MD_nRE cao hơn. Khi các giá trị tăng lên, sự khác biệt tương đối giữa các căn giảm xuống, dẫn đến giá trị MD_nRE thấp hơn.

Dưới đây là một số ưu điểm khi sử dụng MD_nRE làm hàm loss cho các bài toán regression:

- **Robustness to outliers:** Việc lấy căn bậc n trước khi tính loss, đặc biệt cho các trường hợp loss lớn (do outliers) có thể giúp việc optimization của model trở nên dễ dàng và mượt mà hơn.
- **Improved convergence:** Quá trình training nhanh hơn, hội tụ và ổn định hơn khi áp dụng kỹ thuật này.
- **Application domains:** Phương pháp này đặc biệt phù hợp trong các lĩnh vực mà độ lớn của sai số đóng vai trò quan trọng, như tài chính, nơi các sai số nhỏ có thể có tác động tài chính đáng kể.

Phạm vi của bài tập chỉ yêu cầu các bạn viết hàm tính Different of n^{th} Root Error cho một cặp y và $\hat{y}$, và giả sử rằng các điều kiện input đầu vào đều được đáp ứng để đơn giản hoá vấn đề

$$(\sqrt[n]{y} - \sqrt[n]{\hat{y}})^p$$

- Input: y (giá trị của y), y_hat (gía trị của $\hat{y}$), n (căn bậc n), và p (bậc của hàm loss)
- Output: Kết quả của hàm loss

```
md_nre_single_sample(y=100, y_hat=99.5, n=2, p=1)
>> 0.025031328369998107

md_nre_single_sample(y=50, y_hat=49.5, n=2, p=1)
>> 0.03544417213033135

md_nre_single_sample(y=20, y_hat=19.5, n=2, p=1))
>> 0.05625552183565574

md_nre_single_sample(y=0.6, y_hat=0.1, n=2, p=1)
>> 0.45836890322464546
```

Code Listing 6: Đây là các ví dụ các bạn không cần thiết đặt tên giống ví dụ

Các bạn đọc thêm về hàm loss của yolo để thấy được ứng dụng

II. Câu hỏi trắc nghiệm

Câu hỏi 1 : Viết function thực hiện đánh giá classification model bằng F1-Score. Function 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?

```
import math
def calc_f1_score(tp, fp, fn):
    # 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 function **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?

```
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):
    # 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 $\alpha = 0.01$?

```
import math
def calc_elu(x):
    # Your code here

    # End your code

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

a) -1
b) -0.01

c) -0.106321205588285576
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?

```
import math
def calc_activation_func(x, act_name):
    # 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?

```
def calc_ae(y, 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):
  # 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 cos và điều kiện được giới thiệu các bạn click ở đây. Viết function

xấp xỉ cos khi nhận x là giá trị muốn tính cos(x) và n là số lần lặp muốn xấp xỉ. Return 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):
    # 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 10 : Dựa vào công thức xấp xỉ sin và điều kiện được giới thiệu các bạn click ở đây. 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):
    # 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 11 : Dựa vào công thức xấp xỉ sinh và điều kiện được giới thiệu các bạn click ở đây. Viết function xấp xỉ sinh khi nhận x là giá trị muốn tính sinh(x) và n là số lần lặp muốn xấp xỉ. Return 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):
    # 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 các bạn click ở đây. Viết function xấp xỉ cosh khi nhận x là giá trị muốn tính cosh(x) và n là số lần lặp muốn xấp xỉ. Return 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):
    # 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

Câu hỏi 13 : Đoạn code nào thực hiện đúng Mean Difference of n^{th} Root Error được miêu tả ở trên?

```python
# (A)
def md_nre_single_sample(y, y_hat, n, p):
    y_root = y ** (1/n)
    y_hat_root = y_hat ** (1/n)
    difference = y_root - y_hat_root
    loss = difference ** p
    return loss

# (B)
def md_nre_single_sample1(y, y_hat, n, p):
    y_root = y ** (1/n)
    y_hat_root = y_hat ** (1/2)
    difference = y_root - y_hat_root
    loss = difference ** p
    return loss
# (C)
def md_nre_single_sample2(y, y_hat, n, p):
    y_root = y ** (1/n)
    y_hat_root = y_hat ** (1/n)
    difference = y_root / y_hat_root
    loss = difference ** p
    return loss

# (D)
def md_nre_single_sample3(y, y_hat, n, p):
    y_root = y ** (1/n)
    y_hat_root = y_hat ** (1/n)
    difference = y_root - y_hat_root
    loss = difference
    return loss
```

a) (A)
b) (B)
c) (C)
d) (D)