Exp-01.Implementing a Basic Artificial Neuron Model

1.Implementing simple neuron

```
#implementing simple neuron


input1 = 2
input2 = 3
w1= 0.5
w2= 0.8
b = 0.1

sum= (w1*input1)+(w2*input2)+b
print("weightedsum:",sum)

def step_function(x, threshold=0):
    return 1 if x > threshold else 0

output= step_function(sum)

print("output",output)
```

```
weightedsum: 3.5000000000000004
output 1
```

2. Implementing simple neuron using "AND" gate

```
#simple neuron
#using AND gate
def artificial_neuron(inputs,weights,bias):
```

```
    #calculating weighted sum
    weighted_sum=0
    for i in range(len(inputs)):
        weighted_sum+=inputs[i]*weights[i]
    weighted_sum+=bias
    output=1 if  weighted_sum>0 else 0
    return output
inputs=[(0,0),(0,1),(1,0),(1,1)]
weights=[1,1]
bias=-1

for input_combination in inputs:
    output = artificial_neuron(input_combination, weights, bias)
    print(f"Input: {input_combination}, Output: {output}")

#output= artificial_neuron(inputs,weights,bias)
#print("output is :",output)
```

```
Input: (0, 0), Output: 0
Input: (0, 1), Output: 0
Input: (1, 0), Output: 0
Input: (1, 1), Output: 1
```

3. Implementing simple neuron using "AND" gate

```
#simple neuron
#using OR gate
def artificial_neuron(inputs,weights,bias):
    #calculating weighted sum
    weighted_sum=0
    for i in range(len(inputs)):
        weighted_sum+=inputs[i]*weights[i]
    weighted_sum+=bias
    output=1 if  weighted_sum>0 else 0
    return output
inputs=[(0,0),(0,1),(1,0),(1,1)]
weights=[2,2]
bias=-1

for input_combination in inputs:
    output = artificial_neuron(input_combination, weights, bias)
    print(f"Input: {input_combination}, Output: {output}")
```

```
#output= artificial_neuron(inputs,weights,bias)
#print("output is :",output)
```

```
Input: (0, 0), Output: 0
Input: (0, 1), Output: 1
Input: (1, 0), Output: 1
Input: (1, 1), Output: 1
```

4. Implementing simple neuron using "NAND" gate

```
#simple neuron
#using NAND gate
def artificial_neuron(inputs,weights,bias):
    #calculating weighted sum
    weighted_sum=0
    for i in range(len(inputs)):
        weighted_sum+=inputs[i]*weights[i]
    weighted_sum+=bias
    output=1 if  weighted_sum>0 else 0
    return output
inputs=[(0,0),(0,1),(1,0),(1,1)]
weights=[-1,-1]
bias=2

for input_combination in inputs:
    output = artificial_neuron(input_combination, weights, bias)
    print(f"Input: {input_combination}, Output: {output}")

#output= artificial_neuron(inputs,weights,bias)
#print("output is :",output)
```

```
Input: (0, 0), Output: 1
Input: (0, 1), Output: 1
Input: (1, 0), Output: 1
Input: (1, 1), Output: 0
```

5. Implementing simple neuron using "NOR" gate

```python
#simple neuron
#using NOR gate
def artificial_neuron(inputs,weights,bias):
    #calculating weighted sum
    weighted_sum=0
    for i in range(len(inputs)):
        weighted_sum+=inputs[i]*weights[i]
    weighted_sum+=bias
    output=1 if  weighted_sum>0 else 0
    return output
inputs=[(0,0),(0,1),(1,0),(1,1)]
weights=[-2,-2]
bias=1

for input_combination in inputs:
    output = artificial_neuron(input_combination, weights, bias)
    print(f"Input: {input_combination}, Output: {output}")

#output= artificial_neuron(inputs,weights,bias)
#print("output is :",output)
```

```
Input: (0, 0), Output: 1
Input: (0, 1), Output: 0
Input: (1, 0), Output: 0
Input: (1, 1), Output: 0
```

6. Implementing simple neuron using "NOT" gate

```python
# Simple neuron using NOT gate
def artificial_neuron(inputs, weights, bias):
    # Calculating weighted sum
    weighted_sum = 0
    for i in range(len(inputs)):
        weighted_sum += inputs[i] * weights[i]
    weighted_sum += bias
    output = 1 if weighted_sum > 0 else 0
    return output

inputs = [0, 1]
weights = [-3]
bias = 2
```

```
for input_value in inputs:
    output = artificial_neuron([input_value], weights, bias)
    print(f"Input: {input_value}, Output: {output}")
```

```
Input: 0, Output: 1
Input: 1, Output: 0
```

Code cell output actions

7.Spam filtering

```
#spam filtering

import numpy as np

Sample email
email = "There is big discount"

Count capital letters
capital_letter_count = sum(1 for char in email if char.isupper())

Check for spam words
spam_words = ["buy", "cheap", "discount", "offer"]
spam_word_count = sum(1 for word in spam_words if word in email.lower())

Calculate email length
email_length = len(email) if len(email) < 50 else 0

Create feature array
x = np.array([capital_letter_count, email_length, spam_word_count])

Create target output array
y = np.array([1 if spam_word_count > 0 or capital_letter_count > 0 or
email_length > 0 else 0])

Initialize weights and bias
w = np.random.rand(3)
b = np.random.rand(1)

Set learning rate and iterations
learning_rate = 0.0001
iterations = 1000
```

```
Train the model
for i in range(iterations):
    y_predicted = np.dot(x, w) + b
    error = y_predicted - y
    dw = (2 / len(x)) * np.sum(error * x)
    db = (2 / len(x)) * np.sum(error)
    w -= learning_rate * dw
    b -= learning_rate * db

Print final output
print(f"Final weights: {w}")
print(f"Final bias: {b}")
print(f"Final error: {error}")
print(f"Final predicted value: {y_predicted}")
print(f"Final output: {y}")
```

8. Recommendation Systems

```
#recommendation system

import numpy as np

# Simple function to make a recommendation based on age and preference
score
def recommend(age, preference_score, threshold=0.5):
    # Simple logic to combine age and preference score into a single value
(weighted sum)
    score = (age * 0.01) + (preference_score * 0.1)  # Adjust these
coefficients as needed

    # Classify based on threshold
    if score >= threshold:
        return "Yes"  # Recommend Yes
    else:
        return "No"  # Recommend No

# Test the function with example data
test_data = [
    (28, 7),  # Age 28, preference score 7
    (45, 3),  # Age 45, preference score 3
```

```
    (22, 8),  # Age 22, preference score 8
    (32, 5)   # Age 32, preference score 5
]

# Print recommendations for each test case
for age, score in test_data:
    print(f"Age: {age}, Preference Score: {score} -> Recommendation:
{recommend(age, score)}")
```

```
Age: 28, Preference Score: 7 -> Recommendation: Yes
Age: 45, Preference Score: 3 -> Recommendation: Yes
Age: 22, Preference Score: 8 -> Recommendation: Yes
Age: 32, Preference Score: 5 -> Recommendation: Yes
```

9. Binary classification

```
#binary classification

def artificial_neuron(inputs, weights, bias):
    # Calculate the weighted sum (similar to your current code)
    weighted_sum = 0
    for i in range(len(inputs)):
        weighted_sum += inputs[i] * weights[i]
    weighted_sum += bias

    # Apply sigmoid activation (used for binary classification)
    output = 1 / (1 + np.exp(-weighted_sum))  # Sigmoid activation
    return output

# For binary image classification, example inputs might be pixel values of
a processed image
inputs = [0.5, 0.2, 0.8, 0.3]  # These would be the processed pixel values
or feature vectors
weights = [0.6, -0.8, 0.4, 0.1]  # These would be learned during training
bias = -0.1  # Bias also learned during training

# Run through the artificial neuron for one image
output = artificial_neuron(inputs, weights, bias)
if output >= 0.5:
    print("The image contains a cat!")
```

```
else:
    print("The image does not contain a cat!")
```

```
The image contains a cat!
```