Neural Networks & Deep Learning - ICP-3

Name: Akhilesh Chennabatni Mandula
#700 No: 700759335

- **Task 1:**
  - Create a data member to count the number of Employees
  - Create a constructor to initialize name, family, salary, department
  - Create a function to average salary
  - Create a Fulltime Employee class and it should inherit the properties of Employee class
  - Create the instances of Fulltime Employee class and Employee class and call their member functions.
    - This code defines two classes, Employee and FulltimeEmployee, representing employees with various attributes.
    - The Employee class has attributes such as name, family, salary, and department, along with methods to calculate the average salary and display employee details.
    - The FulltimeEmployee class is a subclass of Employee, inheriting its properties.
    - The code then creates instances of both classes, emp1 and emp2 of type Employee, and akhil of type FulltimeEmployee.
    - It initializes their attributes and calls methods to display employee details and the total number of employees.
    - The average salary of all employees is also calculated and printed.
    - The code demonstrates the use of classes, inheritance, object instantiation, and class methods to manage and display employee information.

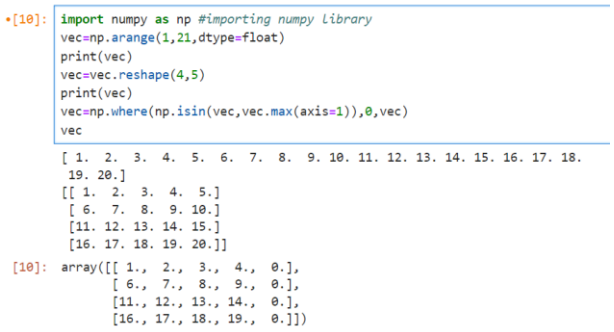
[8]:
```
class Employee:#Employee class
    empCount=0 ##employeeCount is used to keep track of number of employee objects
    salSum=0 #salSum is used to take the average of the salaries of the Employee object
    def __init__(self,name,family,salary,department): #constructor for Employee as per the requirements
        self.name = name
        self.family = family
        self.salary = salary
        self.department = department
        Employee.empCount += 1
        Employee.salSum+=self.salary
    def avg(self):
        return Employee.salSum/Employee.empCount
    @classmethod
    def displayCount(self):
        print("Total Employees:", Employee.empCount)
    def displayEmployee(self):#another function for employee class
        print("Name :", self.name, ", Family:", self.family , ", Salary:", self.salary, ", Department:",self.department)

emp1 = Employee("Akhilesh","CM",2000,"CyberSecurity")#Employee objects creation with intialization
emp2 = Employee("Pooja","Cheekati",5000,"Hardware")
print("avg:",emp2.avg())#calling average function

class FulltimeEmployee(Employee):
    def __init__(self, name, family, salary, department):
        super().__init__(name, family, salary, department)

akhil=FulltimeEmployee("Kaushik","Reddy",9000,"IT")#FultimeEmployee object
akhil.displayEmployee()
emp1.displayCount()
```

```
avg: 3500.0
Name : Kaushik , Family: Reddy , Salary: 9000 , Department: IT
Total Employees: 3
```

- **Task 2:**

  Using NumPy create random vector of size 20 having only float in the range 1-20. Then reshape the array to 4 by 5 Then replace the max in each row by 0 (axis=1) (you can NOT implement it via for loop).

  - Here, a NumPy vector is created using the arange function. It generates an array of evenly spaced values from 1 to 20 (inclusive) with a data type of float.
  - The reshape function is used to change the shape of the vector to a 4x5 matrix. This results in a matrix with 4 rows and 5 columns.
  - vec.max(axis=1) finds the maximum value along each row (axis=1).
  - np.isin(vec, vec.max(axis=1)) checks where each element in the matrix is equal to the maximum value in its corresponding row. This creates a boolean matrix of the same shape as vec.
  - np.where("condition", 0, vec) replaces elements where the condition is True with 0, and leaves other elements unchanged. So, it replaces the maximum values in each row with zero.



```
[10]: import numpy as np #importing numpy library
      vec=np.arange(1,21,dtype=float)
      print(vec)
      vec=vec.reshape(4,5)
      print(vec)
      vec=np.where(np.isin(vec,vec.max(axis=1)),0,vec)
      vec

      [ 1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12. 13. 14. 15. 16. 17. 18.
       19. 20.]
      [[ 1.  2.  3.  4.  5.]
       [ 6.  7.  8.  9. 10.]
       [11. 12. 13. 14. 15.]
       [16. 17. 18. 19. 20.]]
[10]: array([[ 1.,  2.,  3.,  4.,  0.],
             [ 6.,  7.,  8.,  9.,  0.],
             [11., 12., 13., 14.,  0.],
             [16., 17., 18., 19.,  0.]])
```

GitHub Link: https://github.com/Akhil111198/Neural-Networks/tree/08f2916f7ed112170a505e2180719cf48d2e2ab4/ICP3

Video Link: https://drive.google.com/file/d/1q7fKqOep1P0oYHm7YAiI3e83ufYRsCYA/view?usp=sharing

THE END