

Machine Intelligence

Assignment 5

Name : Adarsh Subhas Nayak

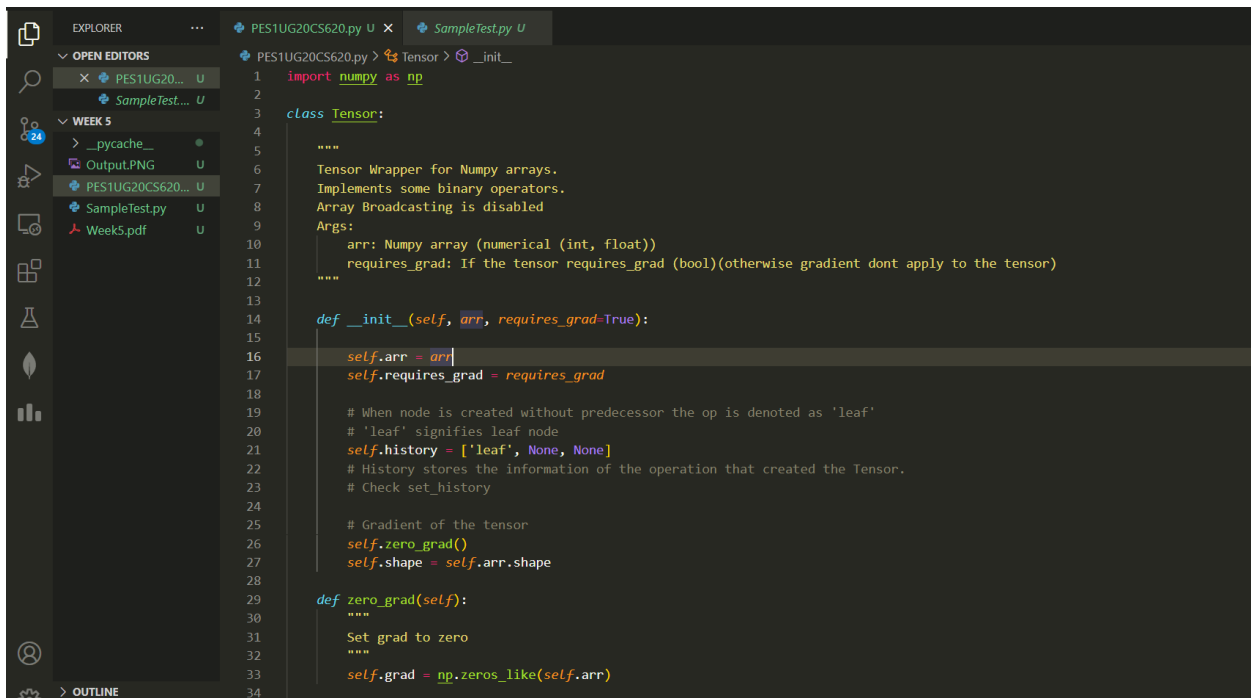
SRN : PES1UG20CS620

Roll No : 54

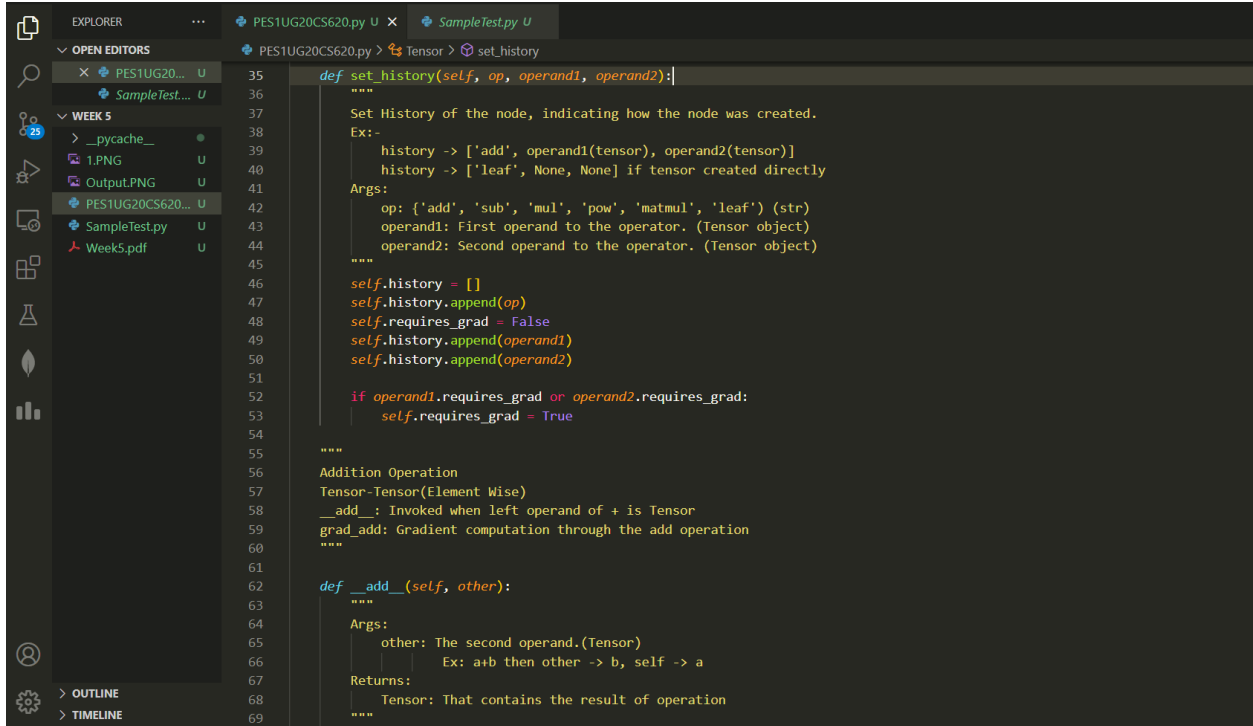
Date : 06-10-2022

Week 5 Assignment :

Code :

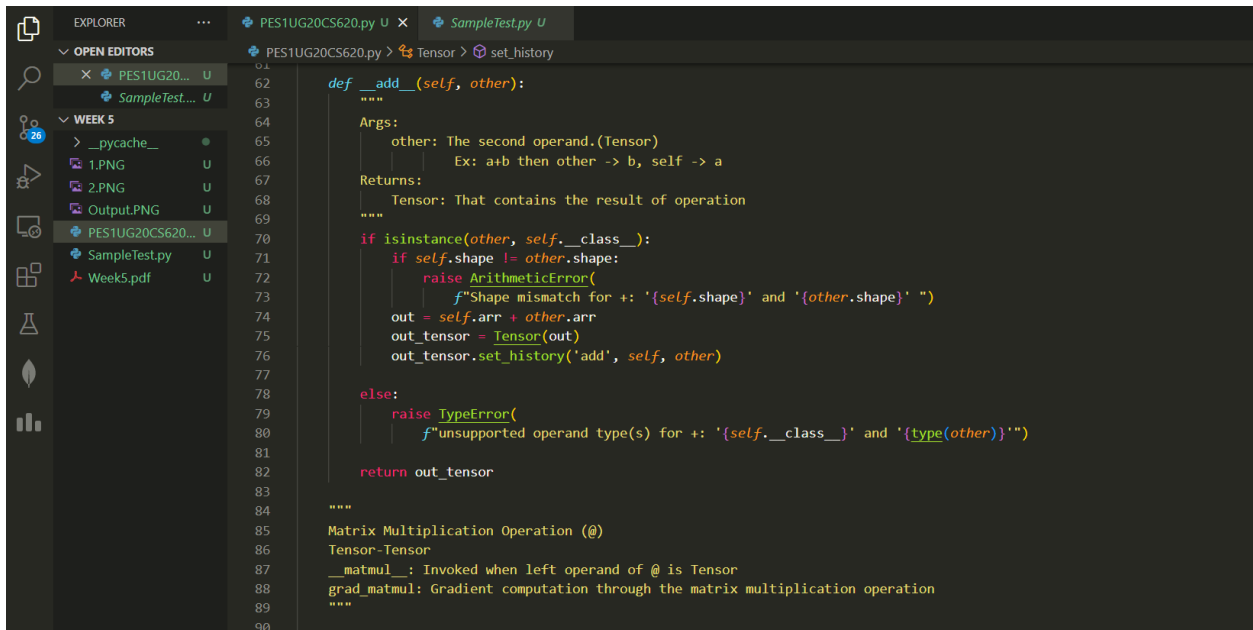


```
1  import numpy as np
2
3  class Tensor:
4
5      """
6      Tensor Wrapper for Numpy arrays.
7      Implements some binary operators.
8      Array Broadcasting is disabled
9      Args:
10         arr: Numpy array (numerical (int, float))
11         requires_grad: If the tensor requires_grad (bool)(otherwise gradient dont apply to the tensor)
12      """
13
14     def __init__(self, arr, requires_grad=True):
15
16         self.arr = arr
17         self.requires_grad = requires_grad
18
19         # When node is created without predecessor the op is denoted as 'leaf'
20         # 'leaf' signifies leaf node
21         self.history = ['leaf', None, None]
22         # History stores the information of the operation that created the Tensor.
23         # Check set_history
24
25         # Gradient of the tensor
26         self.zero_grad()
27         self.shape = self.arr.shape
28
29     def zero_grad(self):
30         """
31         Set grad to zero
32         """
33         self.grad = np.zeros_like(self.arr)
34
```



The screenshot shows the VS Code editor with the Explorer sidebar on the left. The Explorer sidebar shows a file tree with 'OPEN EDITORS' at the top, followed by 'WEEK 5' containing files like '_pycache_', '1.PNG', 'Output.PNG', 'PES1UG20CS620...', 'SampleTest.py', and 'Week5.pdf'. The 'SampleTest.py' file is open in the editor. The editor shows the implementation of the `set_history` method for the `Tensor` class. The method signature is `def set_history(self, op, operand1, operand2):`. The docstring describes the method's purpose: 'Set History of the node, indicating how the node was created. Ex:- history -> ['add', operand1(tensor), operand2(tensor)] history -> ['leaf', None, None] if tensor created directly'. It lists arguments: `op` (operator: 'add', 'sub', 'mul', 'pow', 'matmul', 'leaf'), `operand1` (first operand), and `operand2` (second operand). The implementation sets `self.history` to an empty list, appends the operator, and sets `self.requires_grad` to `False`. It then appends the operands to the history. If either operand requires a gradient, `self.requires_grad` is set to `True`. The docstring also includes: 'Addition Operation', 'Tensor-Tensor(Element Wise)', '`__add__`: Invoked when left operand of + is Tensor', and '`grad_add`: Gradient computation through the add operation'.

```
def set_history(self, op, operand1, operand2):  
    """  
    Set History of the node, indicating how the node was created.  
    Ex:-  
    history -> ['add', operand1(tensor), operand2(tensor)]  
    history -> ['leaf', None, None] if tensor created directly  
    Args:  
        op: {'add', 'sub', 'mul', 'pow', 'matmul', 'leaf'} (str)  
        operand1: First operand to the operator. (Tensor object)  
        operand2: Second operand to the operator. (Tensor object)  
    """  
    self.history = []  
    self.history.append(op)  
    self.requires_grad = False  
    self.history.append(operand1)  
    self.history.append(operand2)  
  
    if operand1.requires_grad or operand2.requires_grad:  
        self.requires_grad = True  
  
    """  
    Addition Operation  
    Tensor-Tensor(Element Wise)  
    __add__: Invoked when left operand of + is Tensor  
    grad_add: Gradient computation through the add operation  
    """  
  
    def __add__(self, other):  
        """  
        Args:  
            other: The second operand.(Tensor)  
            Ex: a+b then other -> b, self -> a  
        Returns:  
            Tensor: That contains the result of operation  
        """
```



The screenshot shows the VS Code editor with the Explorer sidebar on the left. The Explorer sidebar shows a file tree with 'OPEN EDITORS' at the top, followed by 'WEEK 5' containing files like '_pycache_', '1.PNG', '2.PNG', 'Output.PNG', 'PES1UG20CS620...', 'SampleTest.py', and 'Week5.pdf'. The 'SampleTest.py' file is open in the editor. The editor shows the implementation of the `__add__` method for the `Tensor` class. The method signature is `def __add__(self, other):`. The docstring describes the method's purpose: 'Addition Operation (Tensor-Tensor)'. It lists arguments: `other` (the second operand, a Tensor). It lists returns: 'Tensor: That contains the result of operation'. The implementation checks if `other` is an instance of `self.__class__`. If not, it raises an `ArithmeticError` with a message: `f"Shape mismatch for +: '{self.shape}' and '{other.shape}' "`. If it is an instance, it performs the addition: `out = self.arr + other.arr`, creates a new `Tensor` object: `out_tensor = Tensor(out)`, and sets its history: `out_tensor.set_history('add', self, other)`. If `other` is not a `Tensor`, it raises a `TypeError` with a message: `f"unsupported operand type(s) for +: '{self.__class__}' and '{type(other)}'"`. The method returns `out_tensor`. The docstring also includes: 'Matrix Multiplication Operation (@)', 'Tensor-Tensor', '`__matmul__`: Invoked when left operand of @ is Tensor', and '`grad_matmul`: Gradient computation through the matrix multiplication operation'.

```
def __add__(self, other):  
    """  
    Args:  
        other: The second operand.(Tensor)  
        Ex: a+b then other -> b, self -> a  
    Returns:  
        Tensor: That contains the result of operation  
    """  
    if isinstance(other, self.__class__):  
        if self.shape != other.shape:  
            raise ArithmeticError(  
                f"Shape mismatch for +: '{self.shape}' and '{other.shape}' "  
            )  
        out = self.arr + other.arr  
        out_tensor = Tensor(out)  
        out_tensor.set_history('add', self, other)  
    else:  
        raise TypeError(  
            f"unsupported operand type(s) for +: '{self.__class__}' and '{type(other)}'"  
        )  
    return out_tensor  
  
    """  
    Matrix Multiplication Operation (@)  
    Tensor-Tensor  
    __matmul__: Invoked when left operand of @ is Tensor  
    grad_matmul: Gradient computation through the matrix multiplication operation  
    """
```

```
EXPLORER
...
PES1UG20CS620.py X
SampleTest.py U

OPEN EDITORS
PES1UG20... U
SampleTest... U
WEEK 5
> __pycache__
1.PNG U
2.PNG U
3.PNG U
Output.PNG U
PES1UG20CS620... U
SampleTest.py U
Week5.pdf U

OUTLINE
TIMELINE

def __matmul__(self, other):
    """
    Args:
        other: The second operand.(Tensor)
            Ex: a+b then other -> b, self -> a
    Returns:
        Tensor: That contains the result of operation
    """
    if not isinstance(other, self.__class__):
        raise TypeError(
            f"unsupported operand type(s) for matmul: '{self.__class__}' and '{type(other)}'"
        )
    if self.shape[-1] != other.shape[-2]:
        raise ArithmeticError(
            f"Shape mismatch for matmul: '{self.shape}' and '{other.shape}' "
        )
    out = self.arry @ other.arry
    out_tensor = Tensor(out)
    out_tensor.set_history('matmul', self, other)

    return out_tensor

def grad_add(self, gradients=None):
    """
    Find gradients through add operation
    gradients: Gradients from successing operation. (numpy float/int)
    Returns:
        Tuple: (grad1, grad2)
        grad1: Numpy Matrix or Vector(float/int) -> Represents gradients passed to first operand
        grad2: Numpy Matrix or Vector(float/int) -> Represents gradients passed to second operand
        Ex:
            c = a+b
            Gradient to a and b
    """
    # TODO
    op1 = self.history[1]
    op2 = self.history[2]
```

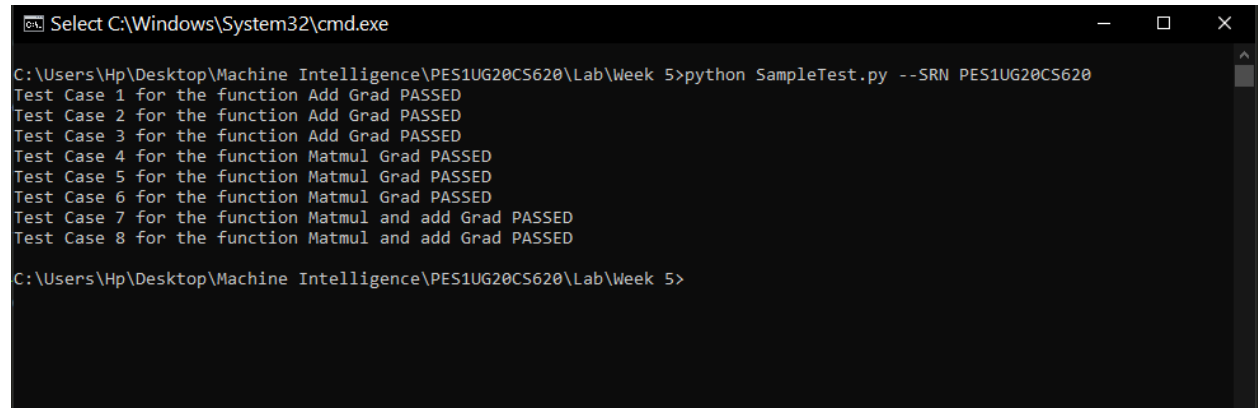
```
EXPLORER  ...  PES1UG20CS620.py X  SampleTest.py U
OPEN EDITORS
  PES1UG20... U 127
  SampleTest... U 128
WEEK 5
  > _pycache_ 130
  1.PNG U 131
  2.PNG U 132
  3.PNG U 133
  4.PNG U 134
  Output.PNG U 135
  PES1UG20CS620... U 136
  SampleTest.py U 137
  Week5.pdf U 138
OUTLINE
TIMELINE

127 op2.grad = np.zeros_like(op2.arr)
128 if op1.requires_grad: op1.grad += np.ones_like(op1.arr)
129 if op2.requires_grad: op2.grad += np.ones_like(op2.arr)
130 if gradients is None:
131     return (op1.grad, op2.grad)
132 if op1.requires_grad: op1.grad = np.multiply(np.ones_like(op1.arr), gradients)
133 if op2.requires_grad: op2.grad = np.multiply(np.ones_like(op2.arr), gradients)
134 return (op1.grad, op2.grad)
135
136 def grad_matmul(self, gradients=None):
137     """
138     Find gradients through matmul operation
139     gradients: Gradients from succeeding operation. (numpy float/int)
140     Returns:
141     Tuple: (grad1, grad2)
142     grad1: Numpy Matrix or Vector(float/int) -> Represents gradients passed to first operand
143     grad2: Numpy Matrix or Vector(float/int) -> Represents gradients passed to second operand
144     Ex:
145     c = a@b
146     Gradients to a and b
147     """
148     # TODO
149     op1 = self.history[1]
150     op2 = self.history[2]
151     if gradients is None:
152         if op1.requires_grad:
153             op1.grad += np.matmul(np.ones_like(op1.arr), op2.arr.transpose())
154         if op2.requires_grad:
155             op2.grad += (np.matmul(np.ones_like(op2.arr), op1.arr)).transpose()
156     else:
157         if op1.requires_grad:
158             op1.grad += np.multiply(np.matmul(np.ones_like(op1.arr), op2.arr.transpose()), gradients)
159         if op2.requires_grad:
160             op2.grad += np.multiply(np.matmul(np.ones_like(op2.arr), op1.arr).transpose(), gradients)
161
```

```
EXPLORER  ...  PES1UG20CS620.py X  SampleTest.py U
OPEN EDITORS
  PES1UG20... U 163
  SampleTest... U 164
WEEK 5
  > _pycache_ 165
  1.PNG U 166
  2.PNG U 167
  3.PNG U 168
  4.PNG U 169
  5.PNG U 170
  Output.PNG U 171
  PES1UG20CS620... U 172
  SampleTest.py U 173
  Week5.pdf U 174
OUTLINE
TIMELINE

163 def backward(self, gradients=None):
164     """
165     Backward Pass until leaf node.
166     Setting the gradient of which is the partial derivative of node(Tensor)
167     the backward in called on wrt to the leaf node(Tensor).
168     Ex:
169     a = Tensor(..) #leaf
170     b = Tensor(..) #leaf
171     c = a+b
172     c.backward()
173     computes:
174     dc/da -> Store in a.grad if a requires_grad
175     dc/db -> Store in b.grad if b requires_grad
176
177     Args:
178     gradients: Gradients passed from succeeding node
179     Returns:
180     Nothing. (The gradients of leaf have to set in their respective attribute(leafobj.grad))
181     """
182     # TODO
183
184     if self.requires_grad == None: return
185     if self.history[0] == 'add':
186
187         gradient = self.grad_add(gradients)
188         if self.history[1]:
189             self.history[1].backward(gradient[0])
190         if self.history[2]:
191             self.history[2].backward(gradient[1])
192
193     elif self.history[0] == 'matmul':
194         gradient = self.grad_matmul(gradients)
195         if self.history[1]:
196             self.history[1].backward(gradient[0])
197         if self.history[2]:
198             self.history[2].backward(gradient[1])
199     else:
200         if self.requires_grad:
201             self.grad = gradients
202
```

Output:



```

Select C:\Windows\System32\cmd.exe

C:\Users\Hp\Desktop\Machine Intelligence\PES1UG20CS620\Lab\Week 5>python SampleTest.py --SRN PES1UG20CS620
Test Case 1 for the function Add Grad PASSED
Test Case 2 for the function Add Grad PASSED
Test Case 3 for the function Add Grad PASSED
Test Case 4 for the function Matmul Grad PASSED
Test Case 5 for the function Matmul Grad PASSED
Test Case 6 for the function Matmul Grad PASSED
Test Case 7 for the function Matmul and add Grad PASSED
Test Case 8 for the function Matmul and add Grad PASSED

C:\Users\Hp\Desktop\Machine Intelligence\PES1UG20CS620\Lab\Week 5>
```