Computer Science Final Year Project

Weekly Progress Report

Week Number: 10

Date: 9th October 2021

Student Reg No:	17B9082	Student Name:	Muhammad Syauqi Waiz bin Haji Sufri
Supervisor Name:	Dr Ahmad Ajaz Bhat		
Project Title:	An episodic memory approach to continual learning systems.		

This week's objectives and progress: (specify experiments' methodology, aim, objectives and results obtained if any)

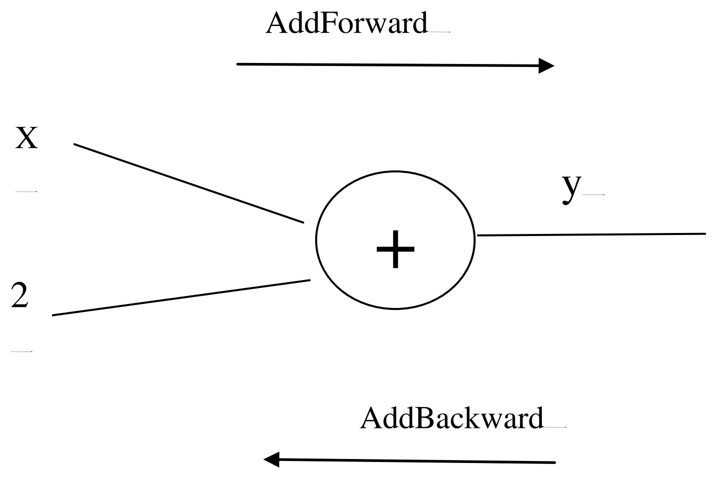
1. Implements at least 1 PyTorch project.
2. Learn and understand EMN codes

1. PyTorch: Finding Gradient

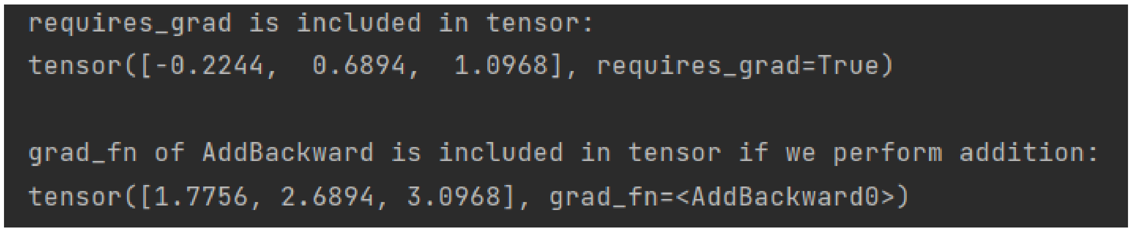
Below is the implementation of code to find gradient using PyTorch

```
import torch

#create a tensor with gradient (requires_grad=False on default)
x = torch.randn(3, requires_grad=True)
print('requires_grad is included in tensor: ')
print(x)
print(" ")

#perform addition on tensors
y = x+2
print("grad_fn of AddBackward is included in tensor if we perform
addition: ")
print(y)
print(" ")
```

Visualization of the addition (y = x + 2):



Output:

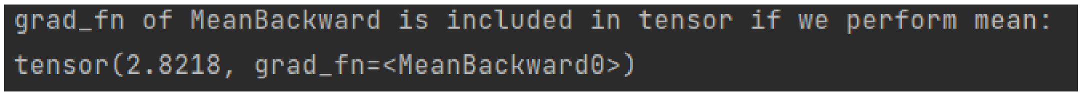


```
requires_grad is included in tensor:
tensor([-0.2244,  0.6894,  1.0968], requires_grad=True)

grad_fn of AddBackward is included in tensor if we perform addition:
tensor([1.7756, 2.6894, 3.0968], grad_fn=<AddBackward0>)
```

Operation can also be substitution, multiplication, division and mean. For example, the mean:

```
#perform multiplication and mean on tensors
z = y*y*2
z= z.mean()
print("grad_fn of MeanBackward is included in tensor if we perform
mean: ")
print(z)
print(" ")
```

Output:

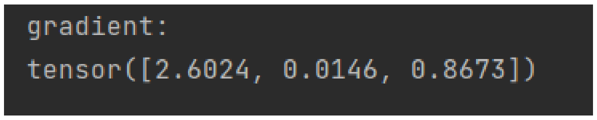


```
grad_fn of MeanBackward is included in tensor if we perform mean:
tensor(2.8218, grad_fn=<MeanBackward0>)
```

Then we can calculate the gradient using the mean since it is a scalar:

```
#Find the gradient using backward() function, it is essentially dz/dx
z.backward()
print("gradient:")
print(x.grad)
print(" ")
```

Output:

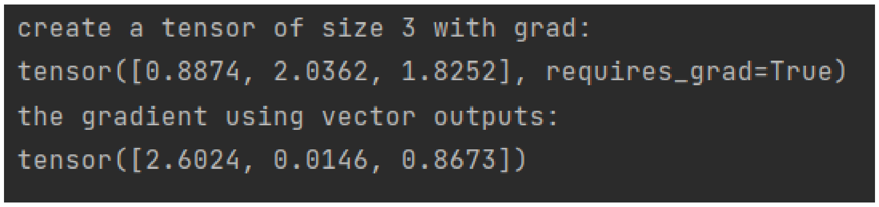


```
gradient:
tensor([2.6024, 0.0146, 0.8673])
```

If the value is not scalar, it cannot be passed to x.grad, hence the operation is different:

```
#x.grad only accept scalar output, therefore for multiple values the
operation is different
print("create a tensor of size 3 with grad:")
a = torch.randn(3, requires_grad=True)
print(a)
b = a+2
#using c as the input
c = b*b*2
#create a tensor with vector of the same size
v = torch.tensor([0.1, 1.0, 0.001], dtype=torch.float32)
#passed an argument on backward()
c.backward(v)
print("the gradient using vector outputs: ")
print(x.grad)
print(" ")
```

Output:

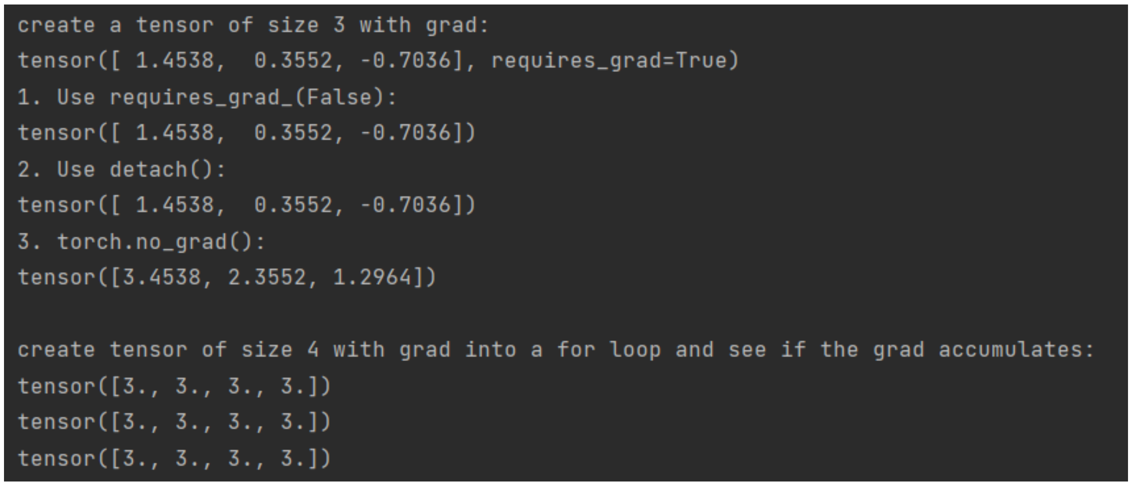


```
create a tensor of size 3 with grad:
tensor([0.8874, 2.0362, 1.8252], requires_grad=True)
the gradient using vector outputs:
tensor([2.6024, 0.0146, 0.8673])
```

Other operations related to finding gradient:

```
#in the case where we do want the gradient to interfere with operation
print("create a tensor of size 3 with grad: ")
d = torch.randn(3, requires_grad=True)
print(d)
#passed any of these 3 arguments
print("1. Use requires_grad_(False): ")
print(d.requires_grad_(False))
print("2. Use detach(): ")
e = d.detach()
print(e)
print("3. torch.no_grad(): ")
with torch.no_grad():
    f = d + 2
    print(f)
print(" ")

#whenever we called backward(), the gradient accumulate in .grad(), so
that needs to change
print("create tensor of size 4 with grad into a for loop and see if
the grad accumulates: ")
weights = torch.rand(4, requires_grad=True)
#create a for loop to observe if it accumulates
for i in range(3):
    model = (weights*3).sum()

    model.backward()

    print(weights.grad)
#use grad.zero_() so that it does not accumulate
    weights.grad.zero_()
```

Output:

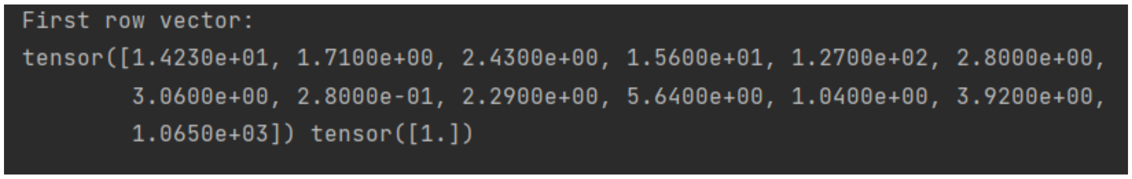


```
create a tensor of size 3 with grad:
tensor([ 1.4538,  0.3552, -0.7036], requires_grad=True)
1. Use requires_grad_(False):
tensor([ 1.4538,  0.3552, -0.7036])
2. Use detach():
tensor([ 1.4538,  0.3552, -0.7036])
3. torch.no_grad():
tensor([3.4538, 2.3552, 1.2964])

create tensor of size 4 with grad into a for loop and see if the grad accumulates:
tensor([3., 3., 3., 3.])
tensor([3., 3., 3., 3.])
tensor([3., 3., 3., 3.])
```

2. Implement Dataset and DataLoader. Both are related to the implementation of EMN.

```
import torch
import torchvision
from torch.utils.data import Dataset, DataLoader
import numpy as np
import math

#create a class for the dataset, in this case I used wine.csv
class WineDataset(Dataset):

#initialization
    def __init__(self):
        #load the csv file
        xy = np.loadtxt('D:/User/Sem Aug 2021/SS-
4290/PyTorch/wine.csv', delimiter=",", dtype=np.float32, skiprows=1)
        #slice the samples starting with the first column
        #convert numpy to tensors
        self.x = torch.from_numpy(xy[:, 1:])
        self.y = torch.from_numpy(xy[:, [0]])
        #get the number of samples
        self.n_samples = xy.shape[0]

#get item function
    def __getitem__(self, item):
        #will return a tuple
        return self.x[item], self.y[item]
#return total no of samples
    def __len__(self):
        return self.n_samples

#constructor
dataset = WineDataset()
#get the first row vector
first_data = dataset[0]
features, labels = first_data
print("First row vector: ")
print(features, labels)
```

Output:



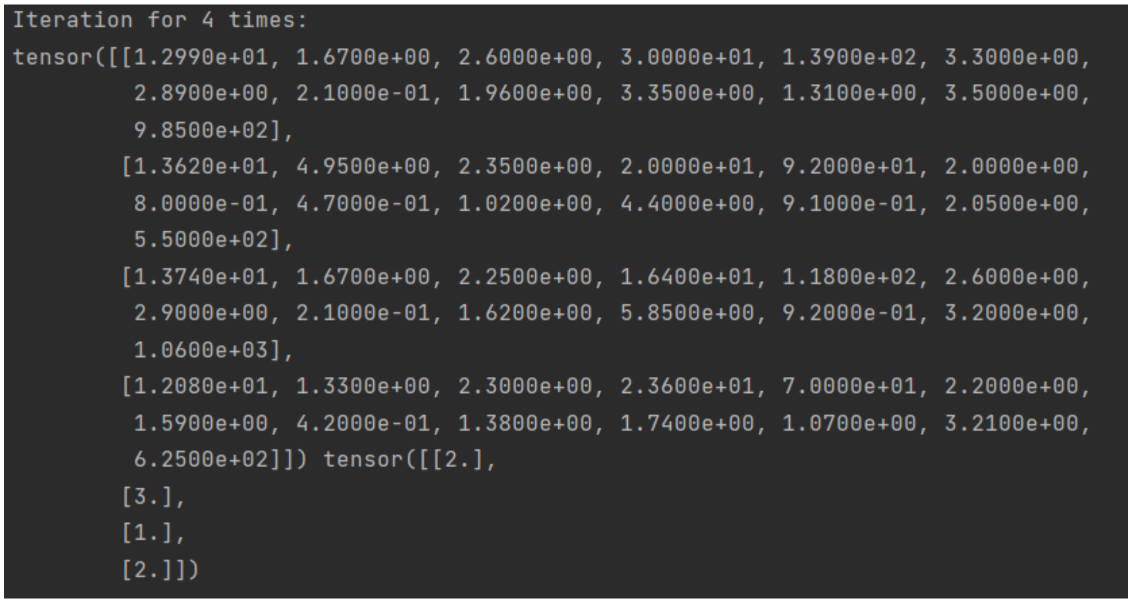
```
First row vector:
tensor([1.4230e+01, 1.7100e+00, 2.4300e+00, 1.5600e+01, 1.2700e+02, 2.8000e+00,
        3.0600e+00, 2.8000e-01, 2.2900e+00, 5.6400e+00, 1.0400e+00, 3.9200e+00,
        1.0650e+03]) tensor([1.])
```

First row with the label at the end.

```
#Using dataloader
dataloader = DataLoader(dataset=dataset, batch_size=4, shuffle=True)

#Perform iteration
datatiter = iter(dataloader)
data = datatiter.next()
features, labels = data
print(features, labels)
```

Using dataloader, initialize dataset with batch_size (no of training samples in forward and backward pass). Perform iteration with iter().

Output:



```
Iteration for 4 times:
tensor([[1.2990e+01, 1.6700e+00, 2.6000e+00, 3.0000e+01, 1.3900e+02, 3.3000e+00,
         2.8900e+00, 2.1000e-01, 1.9600e+00, 3.3500e+00, 1.3100e+00, 3.5000e+00,
         9.8500e+02],
        [1.3620e+01, 4.9500e+00, 2.3500e+00, 2.0000e+01, 9.2000e+01, 2.0000e+00,
         8.0000e-01, 4.7000e-01, 1.0200e+00, 4.4000e+00, 9.1000e-01, 2.0500e+00,
         5.5000e+02],
        [1.3740e+01, 1.6700e+00, 2.2500e+00, 1.6400e+01, 1.1800e+02, 2.6000e+00,
         2.9000e+00, 2.1000e-01, 1.6200e+00, 5.8500e+00, 9.2000e-01, 3.2000e+00,
         1.0600e+03],
        [1.2080e+01, 1.3300e+00, 2.3000e+00, 2.3600e+01, 7.0000e+01, 2.2000e+00,
         1.5900e+00, 4.2000e-01, 1.3800e+00, 1.7400e+00, 1.0700e+00, 3.2100e+00,
         6.2500e+02]]) tensor([[2.],
        [3.],
        [1.],
        [2.]])
```

4 tensors are shown with their labels.

Using for loop to perform iteration:

```
#training loop
#initialiaze epoch to 2
num_epochs = 2
#get total no of samples
total_samples = len(dataset)
#get iteration by dividing total samples with 4(batch_size)
n_iterations = math.ceil(total_samples/4)
#print the total sample and number of iteration per epoch
print("total samples and iteration: ")
print(total_samples, n_iterations)
print(" ")
```

```python
#for loop in the range of no of epoch
for epoch in range(num_epochs):
    for i, (inputs, labels) in enumerate(dataloader):
        #get info every 5 steps
        if(i+1) % 5 == 0:
            print(f'epoch {epoch+1}/{num_epochs}, step
{i+1}/{n_iterations}, inputs {inputs.shape}')
```

Output:

```
total samples and iteration:
178 45

epoch 1/2, step 5/45, inputs torch.Size([4, 13])
epoch 1/2, step 10/45, inputs torch.Size([4, 13])
epoch 1/2, step 15/45, inputs torch.Size([4, 13])
epoch 1/2, step 20/45, inputs torch.Size([4, 13])
epoch 1/2, step 25/45, inputs torch.Size([4, 13])
epoch 1/2, step 30/45, inputs torch.Size([4, 13])
epoch 1/2, step 35/45, inputs torch.Size([4, 13])
epoch 1/2, step 40/45, inputs torch.Size([4, 13])
epoch 1/2, step 45/45, inputs torch.Size([2, 13])
epoch 2/2, step 5/45, inputs torch.Size([4, 13])
epoch 2/2, step 10/45, inputs torch.Size([4, 13])
epoch 2/2, step 15/45, inputs torch.Size([4, 13])
epoch 2/2, step 20/45, inputs torch.Size([4, 13])
epoch 2/2, step 25/45, inputs torch.Size([4, 13])
epoch 2/2, step 30/45, inputs torch.Size([4, 13])
epoch 2/2, step 35/45, inputs torch.Size([4, 13])
epoch 2/2, step 40/45, inputs torch.Size([4, 13])
epoch 2/2, step 45/45, inputs torch.Size([2, 13])
```

End

Overall Progress: (summaries any findings; any achievements, challenges, difficulties encountered)

Findings/Achievements:
1. Able to understand episodic memory.
2. Able to understand memory networks.
3. Able to understand EMN approach to memory-augmented networks.
4. Able to understand MEMO approach to memory-augmented networks.
5. Learn basic of PyTorch

Challenges/Difficulties:
1. Transform my understanding into codes.
2. Coding in general as I may take time to learn how to do perform certain things.
3. COVID-19 partial lockdown affected my work progress (in terms of mental health).

Next week's objectives:

To be decided every weekly meetings.

Date:	9/10/21	Student's Signature:	*SYAUQI*

Supervisor's Comments:-

Date:		Supervisor's Signature:	