Problem 3

In [1]:
```
import torch
import numpy as np
```

In [2]:
```
torch.manual_seed(2139)
z = torch.normal(0, 1 , size=(100, 100), requires_grad=True)
```

In [3]:
```
z.shape
```

Out[3]:
```
torch.Size([100, 100])
```

In [4]:
```
k = 50
```

(a)

In [5]:
```
A = torch.normal(0, 0 , size=(100, k), requires_grad=True)
B = torch.normal(0, 0 , size=(k, 100), requires_grad=True)
```

In [6]:
```
def f(z):
    return A @ B @ z
```

In [7]:
```
 # evaluating data points with Mean Square Error (MSE)
def L(z, fz):
    diff = z - fz
    return 0.5 * (torch.norm(diff, p=2)**2)
```

In [8]:
```
steps = 10
lr = 1e-5

def train(steps, lr, A, B):
    losses = []
    for i in range(steps):
        # Generate Prediction
        fz = f(z)
        # Get the loss and perform backpropagation
        loss = L(z, fz)
        losses.append(loss)
        loss.backward() # get gradient
        # Let's update the weights
        with torch.no_grad():
            A -= lr * A.grad
            B -= lr * B.grad
            # Set the gradients to zero
            A.grad.zero_()
            B.grad.zero_()
#         print(f"step {i}: Loss: {loss}")
    print(f"A==0: ", torch.all(A==0))
    print(f"B==0: ", torch.all(B==0))
    print(f"minimal loss achieved: {min(losses)}")
```

In [9]:
```
train(steps, lr, A, B)
```

```
A==0:  tensor(True)
B==0:  tensor(True)
minimal loss achieved: 4919.1142578125
```

Weights after training is always 0 - because $L(w)$ is always 0 thus results in gradients of 0 which means no update.

(b)

In [10]:
```
A = torch.normal(0, 1/k , size=(100, k), requires_grad=True)
B = torch.normal(0, 1/k , size=(k, 100), requires_grad=True)
```

In [11]:
```
steps = 1000
lrs = [1e-4, 1e-3, 1e-2, 1e-1]

for lr in lrs:
    print("lr: ", lr)
    train(steps, lr, A, B)
```

```
lr:  0.0001
A==0:  tensor(False)
B==0:  tensor(False)
minimal loss achieved: 518.71435546875
lr:  0.001
A==0:  tensor(False)
B==0:  tensor(False)
minimal loss achieved: 506.402587890625
lr:  0.01
A==0:  tensor(False)
B==0:  tensor(False)
minimal loss achieved: 506.3055419921875
lr:  0.1
A==0:  tensor(False)
B==0:  tensor(False)
minimal loss achieved: nan
```

The smallest training error achieved is 506.305 with the learning rate 0.01.

(c) CIFAR10 (optional)

In [12]:
```
import torch
import torchvision
import torchvision.transforms as transforms
```

In [13]:
```
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

batch_size = 1

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size=batch_size,
                                          shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size=batch_size,
                                         shuffle=False, num_workers=2)

classes = ('plane', 'car', 'bird', 'cat',
           'deer', 'dog', 'frog', 'horse', 'ship', 'truck')
```

```
Files already downloaded and verified
Files already downloaded and verified
```

In [14]:
```
# get some random training images
dataiter = iter(testloader)
images, labels = next(dataiter)

images.shape
```

Out[14]:
```
torch.Size([1, 3, 32, 32])
```

In [15]:
```
from tqdm import tqdm
import numpy as np

k=50
img = 3*32*32
A = torch.normal(0, 1/k , size=(img, k), requires_grad=True)
B = torch.normal(0, 1/k , size=(k, img), requires_grad=True)
lr = 1e-4
epoch = 10

def f(z):
    return A @ B @ z

def train(steps, lr, A, B):
    for e in range(epoch):
        losses = []
        print("epoch: ", e)
        for inputs, labels in tqdm(trainloader):
            z = inputs.view(3*32*32)
            # Generate Prediction
            fz = f(z)
            # Get the loss and perform backpropagation
            loss = L(z, fz)
#             print(loss)
            losses.append(loss)
            loss.backward() # get gradient
            # Let's update the weights
            with torch.no_grad():
                A -= lr * A.grad
                B -= lr * B.grad
                # Set the gradients to zero
                A.grad.zero_()
                B.grad.zero_()
        print(f"epoch loss: {np.mean(losses)}")
```

In [16]:
```
train(steps, lr, A, B)
```

```
epoch:  0
  1%||                                              | 301/50000 [00:15<43:10, 19.19i
t/s]
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Input In [16], in <cell line: 1>()
----> 1 train(steps, lr, A, B)

Input In [15], in train(steps, lr, A, B)
     24 #             print(loss)
     25             losses.append(loss)
---> 26             loss.backward() # get gradient
     27             # Let's update the weights
     28             with torch.no_grad():

File ~/anaconda3/lib/python3.9/site-packages/torch/_tensor.py:488, in Tensor
.backward(self, gradient, retain_graph, create_graph, inputs)
    478 if has_torch_function_unary(self):
    479     return handle_torch_function(
    480         Tensor.backward,
    481         (self,),
   (...)
    486         inputs=inputs,
    487     )
--> 488 torch.autograd.backward(
    489     self, gradient, retain_graph, create_graph, inputs=inputs
    490 )

File ~/anaconda3/lib/python3.9/site-packages/torch/autograd/__init__.py:197,
in backward(tensors, grad_tensors, retain_graph, create_graph, grad_variable
s, inputs)
    192     retain_graph = create_graph
    194 # The reason we repeat same the comment below is that
    195 # some Python versions print out the first line of a multi-line func
tion
    196 # calls in the traceback and some print out the last line
--> 197 Variable._execution_engine.run_backward(  # Calls into the C++ engin
e to run the backward pass
    198     tensors, grad_tensors_, retain_graph, create_graph, inputs,
    199     allow_unreachable=True, accumulate_grad=True)

KeyboardInterrupt:
```

Sorry, it took so long time on cpu so give up at this point..