```
### Cosc4337 Fall 2022 Rizk - HW5

# Dosbol Aliev
# MyUH 1867424
# Data SCience II

# For this homework, I have used Google Collab, so It was no need to install Libraries

import torch
from torch import nn

import math
import matplotlib.pyplot as plt
import torchvision
import torchvision.transforms as transforms

torch.manual_seed(111)
```

```
<torch._C.Generator at 0x7f28631c07f0>
```

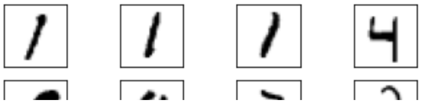
```
print(f"Is CUDA supported by this system? {torch.cuda.is_available()}")
```

```
Is CUDA supported by this system? True
```

```
device = ""
if torch.cuda.is_available():
    device = torch.device("cuda")
else:
    device = torch.device("cpu")
print(device)
```

```
cuda
```

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

train_set = torchvision.datasets.MNIST(
    root=".", train=True, download=True, transform=transform
)

batch_size = 32
train_loader = torch.utils.data.DataLoader(
    train_set, batch_size=batch_size, shuffle=True
)

real_samples, mnist_labels = next(iter(train_loader))
for i in range(16):
    ax = plt.subplot(4, 4, i + 1)
    plt.imshow(real_samples[i].reshape(28, 28), cmap="gray_r")
    plt.xticks([])
    plt.yticks([])
```



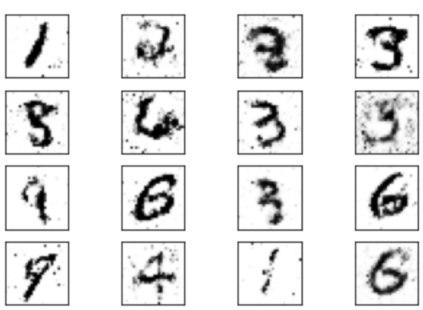
```
class Discriminator(nn.Module):
    def __init__(self):
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(784, 1024),
            nn.ReLU(),
            nn.Dropout(0.5), # Making DropOut Rate 50%
            nn.Linear(1024, 512),
            nn.ReLU(),
            nn.Dropout(0.5), # Making DropOut Rate 50%
            nn.Linear(512, 256),
            nn.ReLU(),
            nn.Dropout(0.5), # Making DropOut Rate 50%
            nn.Linear(256, 1),
            nn.Sigmoid(),
        )

    def forward(self, x):
        x = x.view(x.size(0), 784)
        output = self.model(x)
        return output
discriminator=Discriminator().to(device=device)


class Generator(nn.Module):
    def __init__(self):
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(100, 256),
            nn.ReLU(),
            nn.Linear(256, 512),
            nn.ReLU(),
            nn.Linear(512, 1024),
            nn.ReLU(),
            nn.Linear(1024, 784),
            nn.Tanh(),
        )

    def forward(self, x):
        output = self.model(x)
        output = output.view(x.size(0), 1, 28, 28)
        return output
generator=Generator().to(device=device)
```

```
lr = 1 * (10)**(-4)
num_epochs = 50 # Making number of Epochs 50 because in pdf file also 50
loss_function = nn.BCELoss()

optimizer_discriminator = torch.optim.Adam(discriminator.parameters(), lr=lr)
optimizer_generator = torch.optim.Adam(generator.parameters(), lr=lr)


for epoch in range(num_epochs):
    for n,(real_samples,mnist_labels) in enumerate(train_loader):

        # Data for training the discriminator
        real_samples=real_samples.to(device=device)
        real_sample_labels=torch.ones((batch_size,1)).to(device=device)
```

```
        latent_space_samples=torch.randn((batch_size,100)).to(device=device)


        generated_samples=generator(latent_space_samples)


        generated_sample_labels=torch.zeros((batch_size,1)).to(device=device)


        all_samples=torch.cat((real_samples,generated_samples))
        all_sample_labels=torch.cat((real_sample_labels,generated_sample_labels))

        # Training the discriminator


        discriminator.zero_grad()
        output_discriminator=discriminator(all_samples)
        loss_discriminator=loss_function(output_discriminator,all_sample_labels)
        loss_discriminator.backward()
        optimizer_discriminator.step()



        latent_space_samples=torch.randn((batch_size,100)).to(device=device)



        generator.zero_grad()
        generated_samples=generator(latent_space_samples)
        output_discriminator_generated=discriminator(generated_samples)
        loss_generator=loss_function(output_discriminator_generated,real_sample_labels)
        loss_generator.backward()
        optimizer_generator.step()

         # Show loss
        if n==batch_size-1:
            print(f"Epoch: {epoch} Loss D.: {loss_discriminator}")
            print(f"Epoch: {epoch} Loss G.: {loss_generator}")
            plt.show(epoch,loss_discriminator)
            plt.show(epoch,loss_generator)
```

```
Epoch: 34 Loss D.: 0.6550514698028564
Epoch: 34 Loss G.: 0.922463595867157
Epoch: 35 Loss D.: 0.6006709337234497
Epoch: 35 Loss G.: 0.9175853729248047
Epoch: 36 Loss D.: 0.6752769351005554
Epoch: 36 Loss G.: 0.786282479763031
Epoch: 37 Loss D.: 0.6443865895271301
Epoch: 37 Loss G.: 0.8613917231559753
Epoch: 38 Loss D.: 0.6414182782173157
Epoch: 38 Loss G.: 0.8122552633285522
Epoch: 39 Loss D.: 0.5926834940910339
Epoch: 39 Loss G.: 0.8782569766044617
Epoch: 40 Loss D.: 0.6065706014633179
Epoch: 40 Loss G.: 0.924579381942749
Epoch: 41 Loss D.: 0.6939267516136169
Epoch: 41 Loss G.: 0.8911184072494507
Epoch: 42 Loss D.: 0.6493178009986877
Epoch: 42 Loss G.: 0.9285207986831665
Epoch: 43 Loss D.: 0.6092119216918945
Epoch: 43 Loss G.: 0.9139573574066162
Epoch: 44 Loss D.: 0.5990184545516968
Epoch: 44 Loss G.: 0.9169609546661377
Epoch: 45 Loss D.: 0.630357027053833
Epoch: 45 Loss G.: 0.7320483922958374
Epoch: 46 Loss D.: 0.663044810295105
Epoch: 46 Loss G.: 0.862614631652832
Epoch: 47 Loss D.: 0.6141297817230225
Epoch: 47 Loss G.: 0.8665668964385986
Epoch: 48 Loss D.: 0.6164787411689758
Epoch: 48 Loss G.: 0.8401762247085571
Epoch: 49 Loss D.: 0.6607376337051392
Epoch: 49 Loss G.: 0.8308765292167664
```

```
latent_space_samples = torch.randn(batch_size, 100).to(device=device)
generated_samples = generator(latent_space_samples)

# Make sure your output is something like what is below and you should be good.

generated_samples = generated_samples.cpu().detach()
for i in range(16):
    ax = plt.subplot(4, 4, i + 1)
    plt.imshow(generated_samples[i].reshape(28, 28), cmap="gray_r")
    plt.xticks([])
    plt.yticks([])
```



✓ 0s completed at 9:23 PM