CS561 HW14

December 9, 2023

1 Problem 1

We define the entropy of continuous random variable X as follows:

$$H(X) = \mathbb{E}_X[-\log(p(x))] = \int_{\mathcal{X}} -p(x)\log(p(x))dx$$

a) Let $\mathbf{x} \in \mathbb{R}^n$ be a random vector, be a constant vector, and define $\mathbf{y} = \mathbf{x} +$. We can compute the entropy of $\mathbf{y}$ as follows:

$$\begin{aligned}
H(\mathbf{y}) &= \mathbb{E}_X[-\log(p(\mathbf{x} + \ ))] \\
&= \int_{\mathbb{R}^n} -p(\mathbf{x} + \ )\log(p(\mathbf{x} + \ ))d\mathbf{x} \\
&= \int_{\mathbb{R}^n} -p(\mathbf{x} + \ )\log(p(\mathbf{x} + \ ))d(x + \mu) \\
&= \int_{\mathbb{R}^n} -p(\mathbf{u})\log(p(\mathbf{u}))d\mathbf{u} \\
&= H(\mathbf{x})
\end{aligned}$$

b) Let $\mathbf{x} \in \mathcal{N}(0, \Sigma)$, we can compute the entropy of $\mathbf{x}$ as follows:

$$\begin{aligned}
H(\mathbf{x}) &= -\mathbb{E}[\log(p(\mathbf{x}))] \\
&= \frac{1}{2}\mathbb{E}\left[\log\left((2\pi)^n|\Sigma|\right) + \mathbf{x}\Sigma^{-1}\mathbf{x}^T\right] \\
&= \frac{\log\left((2\pi)^n|\Sigma|\right)}{2} + \frac{1}{2}\mathbb{E}\left[\mathbf{x}\Sigma^{-1}\mathbf{x}^T\right] \\
&= \frac{n\log(\pi)}{2} + \frac{|\Sigma|}{2} + \frac{1}{2}\mathbb{E}\left[\mathrm{tr}(\mathbf{x}\Sigma^{-1}\mathbf{x}^T)\right] \\
&= \frac{n\log(\pi)}{2} + \frac{|\Sigma|}{2} + \frac{1}{2}\mathbb{E}\left[\mathrm{tr}(\Sigma^{-1}\mathbf{x}^T\mathbf{x})\right] \\
&= \frac{n\log(\pi)}{2} + \frac{|\Sigma|}{2} + \frac{1}{2}\mathrm{tr}(\Sigma^{-1}\mathbb{E}\left[\mathbf{x}^T\mathbf{x}\right]) \\
&= \frac{n\log(\pi)}{2} + \frac{|\Sigma|}{2} + \frac{1}{2}\mathrm{tr}(\Sigma^{-1}\Sigma) \\
&= \frac{n\log(\pi)}{2} + \frac{|\Sigma|}{2} + \frac{n}{2}
\end{aligned}$$

when n is the dimension of the multivariate gaussian in question.

c) Adding a constant mean does not affect the differential entropy of a random variable. Therefore, $H(\mathcal{N}(\mu, \Sigma)) = \frac{n\log(\pi)}{2} + \frac{|\Sigma|}{2} + \frac{n}{2}$.

2 Problem 2

VAE implementation by me (KK Thuwajit / Kuroma). Code resources: my old github repository (https://github.com/konkuad/GANime) from several years ago.

[1]:
```
#Imports
import numpy as np
import matplotlib.pyplot as plt
import torch
from torch import nn
import random
from tqdm import tqdm
import torchvision
import torchvision.datasets as datasets

#for consistency, all seeds are set to 69420
seed = 69420
random.seed(seed)
np.random.seed(seed)
torch.manual_seed(seed)
torch.cuda.manual_seed(seed)
```

2.1 2.1) VAE Encoder

This model attempts to transform real world data to $\mathcal{N}(0, I)$. It has three components: the convolutional encoder that maps images to a 512-dimensional vector, a fully connected layer for the predicted means, and another one for covariance. We use the reparameterization trick (i.e. https://www.baeldung.com/cs/vae-reparameterization) to randomly sample from latent space by shifting/scaling a randomly sampled standard normal vector by our computed mean and variances. The model keeps track of its KL-divergence to be backpropagated at training time.

[2]:
```
class Encoder(nn.Module):
    def __init__(self, latent_size, img_channel):
        super(Encoder, self).__init__()

        self.encoder = nn.Sequential(
            nn.Conv2d(img_channel, 32, 4, 2, 1, bias=False),
            nn.BatchNorm2d(32),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(32, 64, 4, 2, 1, bias=False),
            nn.BatchNorm2d(64),
            nn.LeakyReLU(0.2, inplace=True),

            nn.Conv2d(64, 128, 4, 2, 1, bias=False),
            nn.BatchNorm2d(128),
            nn.LeakyReLU(0.2, inplace=True),
```

```
            nn.Conv2d(128, 256, 4, 2, 1, bias=False),
            nn.BatchNorm2d(256),
            nn.LeakyReLU(0.2, inplace=True),
        )
        self.mu_fc = nn.Conv2d(256, latent_size, 2, 1, 0, bias=False)
        self.sigma_fc = nn.Conv2d(256, latent_size, 2, 1, 0, bias=False)
        self.kl = 0

    def forward(self,x,normal_generator):
        encoded = self.encoder(x)
        mu = self.mu_fc(encoded).mean([-2,-1]) #mean
        sigma = torch.exp(self.sigma_fc(encoded).mean([-2,-1])) #covariance
        z = mu + sigma*normal_generator.sample(mu.shape) #randomly sample from␣
↪latent
        self.kl = (sigma**2 + mu**2 - torch.log(sigma) - 1/2).sum() # kl loss␣
↪term
        return z
```

2.2 2.2) VAE Decoder

This convolutional neural network model maps latent vectors and reconstruct images from it. This decoder is objectively trained using the L2 reconstruction loss (though L1 would work under a different maximum likelihood assumption, as discussed in class).

[3]:
```
class Decoder(nn.Module):

    def __init__(self, latent_size, img_channel):

        super(Decoder, self).__init__()

        self.conv_transpose_block_1 = nn.Sequential(
            nn.ConvTranspose2d(latent_size, 256, 4, 1, 0, bias=False),
            nn.BatchNorm2d(256),
            nn.LeakyReLU(0.2, inplace=True))

        self.conv_transpose_block_2 = nn.Sequential(
            nn.ConvTranspose2d(256, 128, 4, 2, 1, bias=False),
            nn.BatchNorm2d(128),
            nn.LeakyReLU(0.2, inplace=True))

        self.conv_transpose_block_3 = nn.Sequential(
            nn.ConvTranspose2d(128, 64, 4, 2, 1, bias=False),
            nn.BatchNorm2d(64),
            nn.LeakyReLU(0.2, inplace=True))

        self.conv_transpose_block_4 = nn.Sequential(
            nn.ConvTranspose2d(64, 32, 4, 2, 1, bias=False),
```

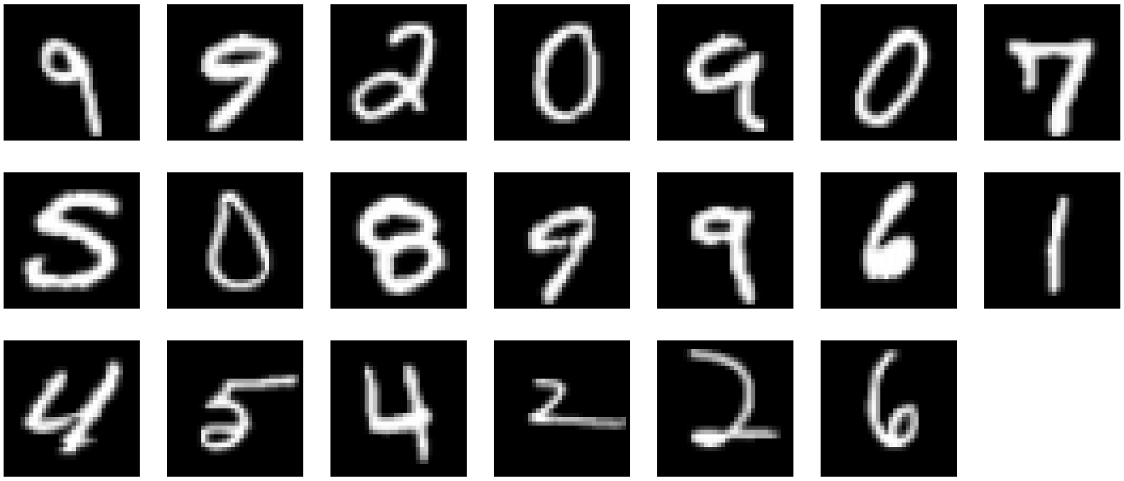
```
            nn.BatchNorm2d(32),
            nn.LeakyReLU(0.2, inplace=True))

        self.conv_transpose_block_5 = nn.Sequential(
            nn.ConvTranspose2d(32, img_channel, 1, 1, 0, bias=False),
            nn.Sigmoid())

    def forward(self, x):
        x = self.conv_transpose_block_1(x)
        x = self.conv_transpose_block_2(x)
        x = self.conv_transpose_block_3(x)
        x = self.conv_transpose_block_4(x)
        x = self.conv_transpose_block_5(x)
        return x
```

2.3 2.3) Dataloader

We load the MNIST dataset using torchvision's built-in loader. We normalize each pixel to (0,1) and resize to 32x32 for shape convenience.

[4]:
```
transforms = torchvision.transforms.Compose([
    torchvision.transforms.Resize((32,32)),
    torchvision.transforms.ToTensor()
])

mnist_trainset = datasets.MNIST(root='./data', train=True, download=True,␣
 ↪transform=transforms)
data_loader = torch.utils.data.
 ↪DataLoader(mnist_trainset,batch_size=128,shuffle=True,num_workers=1)

plt.gray()

def plotter(images):
    f = -(-len(images)//3)
    fig,ax = plt.subplots(3,f,figsize=(5*f, 15))
    for i in range(len(images)):
        ax[i%3,i//3].imshow(images[i,0])
    for aa in ax:
        for aaa in aa:
            aaa.axis("off")
    plt.show()

for a, b in data_loader:
    plotter(a[0:20])
    break
```

```
<Figure size 640x480 with 0 Axes>
```



2.4 2.4) Training

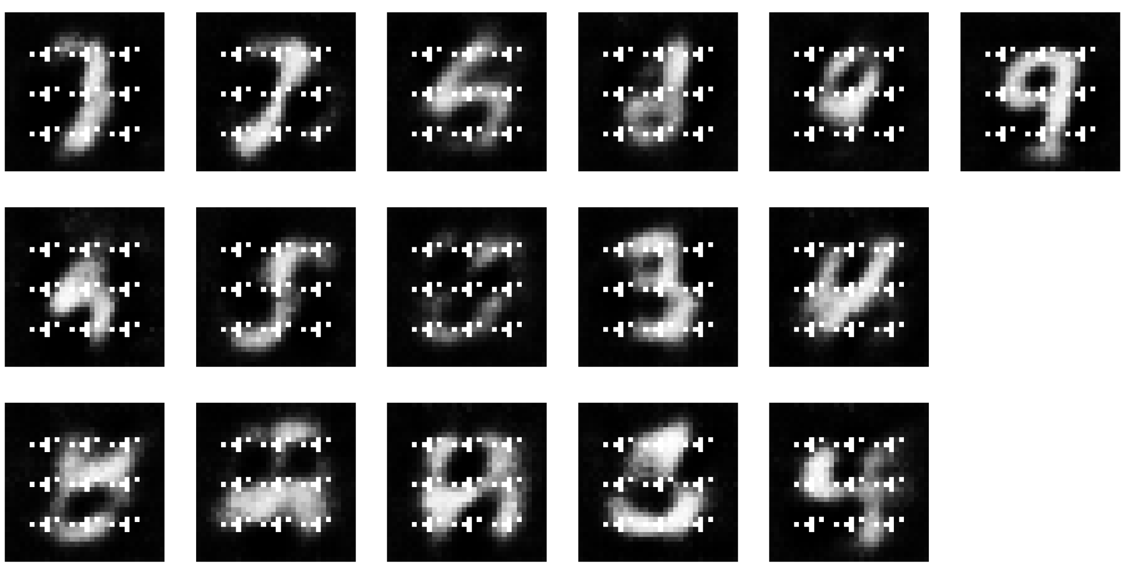
We train the VAE with two losses: KL divergence and reconstruction (L2)

[5]:
```
latent_size = 128
img_ch = 1
num_epochs = 20

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
normal_generator = torch.distributions.Normal(torch.tensor(0).float().
 ↪to(device), torch.tensor(1).float().to(device))

enc = Encoder(latent_size, img_ch).to(device)
dec = Decoder(latent_size, img_ch).to(device)

optimizer = torch.optim.Adam(list(enc.parameters())+list(dec.parameters()),␣
 ↪lr=0.0002, betas=(0.5, 0.999))
```

[6]:
```
visualize_noise = torch.randn(16, latent_size, 1, 1).float().to(device)

for epoch in range(num_epochs):
    pbar = tqdm(enumerate(data_loader))
    enc.train()
    dec.train()
    count = 0
    kl_sum = 0
    l2_sum = 0
    for i, (data, _) in pbar:
        optimizer.zero_grad()
        x = data.to(device)
        b = x.shape[0]
```

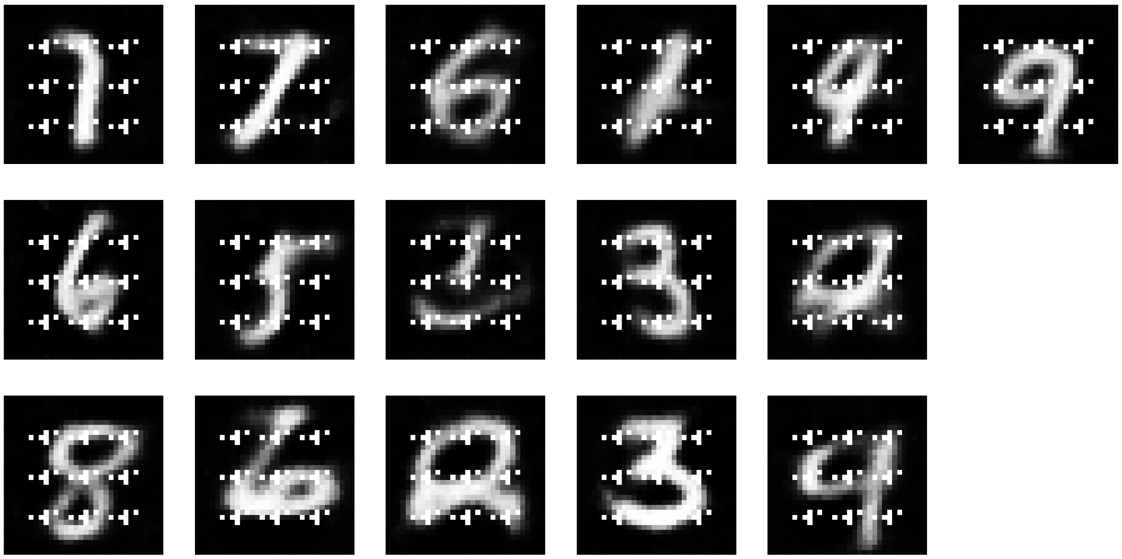
```
        z = enc(x, normal_generator)
        kl_loss = enc.kl
        reconstructed = dec(z.reshape(b,latent_size,1,1))
        l2_loss = ((reconstructed - x)**2).sum()
        vae_loss = kl_loss + l2_loss
        vae_loss.backward()
        optimizer.step()

        count += b
        kl_sum += kl_loss.item()
        l2_sum += l2_loss.item()

        kl_loss_show = '{:.4f}'.format(kl_sum/count)
        l2_loss_show = '{:.4f}'.format(l2_sum/count)
        pbar.set_description(f'Iteration {i+1}/{len(data_loader)}\t[Epoch {epoch+1}/{num_epochs}]\tLosses:\tKL = {kl_loss_show} \tL2 = {l2_loss_show}')

    # visualize every 4 epochs
    if epoch%4==0:
        with torch.no_grad():
            dec.eval()
            pred = dec(visualize_noise.to(device).float())
            plotter(pred.cpu())
```

```
Iteration 469/469       [Epoch 1/20]    Losses: KL = 54.6165    L2 = 74.8870: :
469it [00:16, 28.60it/s]
```

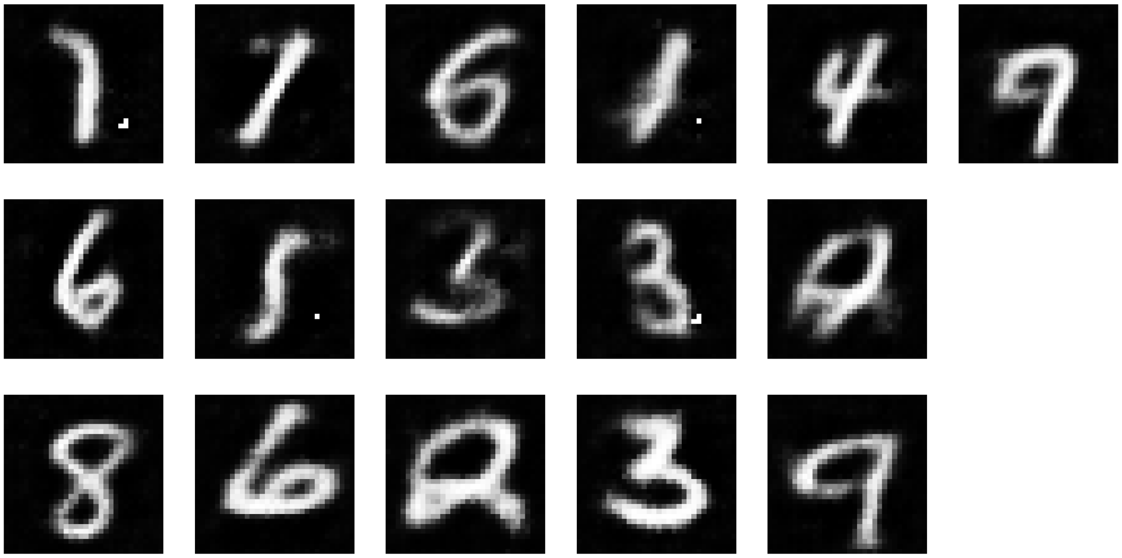


```
Iteration 469/469       [Epoch 2/20]    Losses: KL = 55.1004    L2 = 59.6094: :
469it [00:06, 69.10it/s]
```

```
Iteration 469/469       [Epoch 3/20]    Losses: KL = 55.4111    L2 = 57.1922: :
469it [00:07, 64.88it/s]
Iteration 469/469       [Epoch 4/20]    Losses: KL = 55.5908    L2 = 55.7527: :
469it [00:07, 65.09it/s]
Iteration 469/469       [Epoch 5/20]    Losses: KL = 55.7676    L2 = 54.7117: :
469it [00:07, 63.80it/s]
```



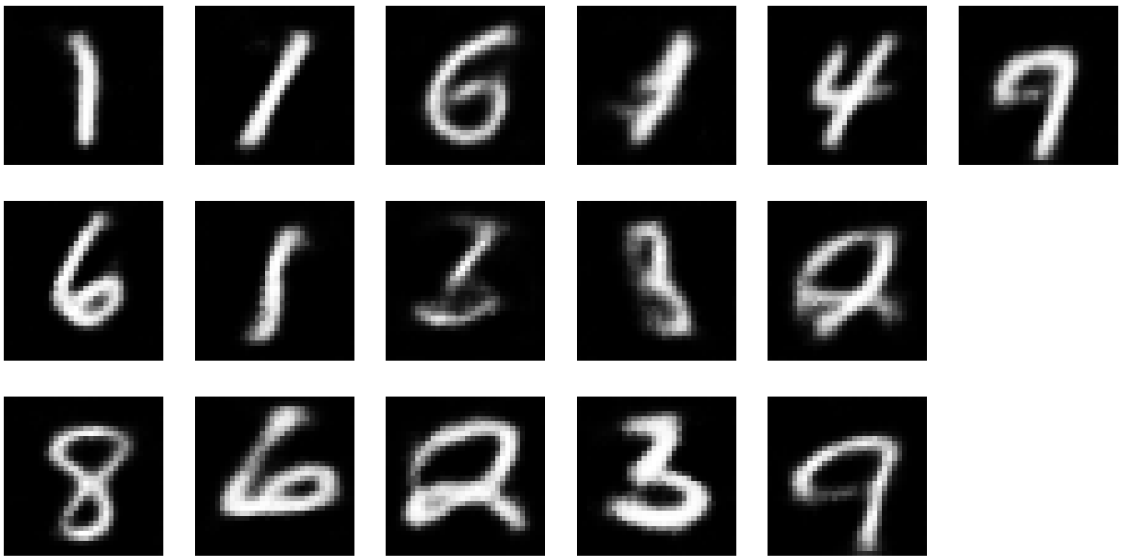
```
Iteration 469/469       [Epoch 6/20]    Losses: KL = 55.8834    L2 = 53.9814: :
469it [00:07, 64.22it/s]
Iteration 469/469       [Epoch 7/20]    Losses: KL = 55.9864    L2 = 53.4166: :
469it [00:07, 61.57it/s]
Iteration 469/469       [Epoch 8/20]    Losses: KL = 56.0832    L2 = 51.3725: :
469it [00:06, 68.87it/s]
Iteration 469/469       [Epoch 9/20]    Losses: KL = 57.0602    L2 = 19.0327: :
469it [00:06, 74.65it/s]
```



```
Iteration 469/469       [Epoch 10/20]   Losses: KL = 56.9860    L2 = 16.4705: :
469it [00:06, 68.95it/s]
Iteration 469/469       [Epoch 11/20]   Losses: KL = 57.0425    L2 = 15.7920: :
469it [00:07, 66.48it/s]
Iteration 469/469       [Epoch 12/20]   Losses: KL = 57.0736    L2 = 15.4482: :
469it [00:06, 68.90it/s]
Iteration 469/469       [Epoch 13/20]   Losses: KL = 57.1086    L2 = 15.2128: :
469it [00:07, 65.75it/s]
```

```
Iteration 469/469       [Epoch 14/20]   Losses: KL = 57.1252    L2 = 14.9743: :
```

```
469it [00:06, 67.81it/s]
Iteration 469/469       [Epoch 15/20]   Losses: KL = 57.1699    L2 = 14.7871: :
469it [00:07, 63.17it/s]
Iteration 469/469       [Epoch 16/20]   Losses: KL = 57.1752    L2 = 14.6092: :
469it [00:07, 65.41it/s]
Iteration 469/469       [Epoch 17/20]   Losses: KL = 57.1972    L2 = 14.4964: :
469it [00:07, 66.84it/s]
```



```
Iteration 469/469       [Epoch 18/20]   Losses: KL = 57.2064    L2 = 14.3362: :
469it [00:07, 64.35it/s]
Iteration 469/469       [Epoch 19/20]   Losses: KL = 57.2227    L2 = 14.1998: :
469it [00:07, 66.23it/s]
Iteration 469/469       [Epoch 20/20]   Losses: KL = 57.2319    L2 = 14.0880: :
469it [00:07, 65.52it/s]
```

2.5 2.5) Random generation

We randomly generate images using our finished VAE. Observation: VAE's use of L2 to approximate its variational lower bound causes its generations to possess soft, undefined edges (unlike GANs).

[7]:
```python
with torch.no_grad():
    dec.eval()
    pred = dec(visualize_noise.to(device).float())
    plotter(pred.cpu())
```