Hw13

周亦涵 2020012853 未央-水木01

Problem1

If the training objective doesn't contain the second regularization term, that is we want to maximize only:

$$L_{new}(x,\theta,\phi) = E_{z\sim q_\phi(z|x)}[\log p_\theta(x|z)] \tag{1}$$

Here, the encoder encodes the input data into a distribution of the latent space, suppose it's Gaussian:

$$q_\phi(z|x) = N(z;\mu_\phi(x),\sigma^2_\phi(x)) \tag{2}$$

The decoder maps any latent code to a meaningful data distribution, suppose it's Gaussian too and has fixed variance:

$$p_\theta(x|z) = N(x;\mu_\theta(z),\sigma^2) \tag{3}$$

Encoder and decoder are jointly trained to maximize the new lower bound:

$$\theta^*,\phi^* = \arg\max_{\theta,\phi} \Sigma_{i=1}^N L_{new}(x_i,\theta,\phi) \tag{4}$$

The likelihood function:

$$\log p_\theta(x|z) = -\frac{d}{2}\log\sigma^2 - \frac{||x-\mu_\theta(z)||^2}{2\sigma^2} \tag{5}$$

So the reconstruction loss: (C as a constant that doesn't affect the optimization process)

$$E_{z\sim q_\phi(z|x)}[\log p_\theta(x|z)] \approx \frac{1}{K}\Sigma_{k=1}^K \log p_\theta(x|z^{(k)}) \tag{6}$$

$$= -\frac{1}{2\sigma^2 K}\Sigma_{k=1}^K ||x-\mu_\theta(z)||^2 + C \tag{7}$$

During the training process, for the forward encoder, we first introduce a noice $\epsilon \sim N(0,1)$, then sample $z^{(k)} = \mu_\phi(x) + \sigma_\phi(x)\cdot\epsilon^{(k)}$, so that we have $z\sim q_\phi(z|x)$. Since we don't have the regularization term, we don't need to compute L_{reg} this time. For the forward decoder, we compute reconstruction term:

sample $x' \sim p_\theta(x|z^{(k)})$, then $L_{recon} = -\frac{1}{2\sigma^2 K}||x - x'||^2$, and we maximize $L = L_{recon}$ with gradient ascent.

As a result, the encoder will solely focus on maximizing the likelihood of the data, so that it would only try to capture the most probable latent space for a given input rather than learning a meaningful and structured latent space. As a result, the latent representations will be less informative. The decoder will also focus on maximizing the log likelihood and could lead to overfitting, it would closely resemble the training data instead of the true underlying distribution in the latent space. The sample generation will be overconfitent.

Problem2

The realization of `GAN` on `MNIST` dataset.

The code is as following:

```
# 1. Prepare D network's gradient
self.D_optimizer.zero_grad()

# 2. Calculate the loss for self.D in real images, label is self.y_real_
D_real = self.D(x_)
D_real_loss = self.BCE_loss(D_real,self.y_real_)

# 3. Generate the fake images, let self.D classify it, using label
self.y_fake_
G_ = self.G(z_)
D_fake = self.D(G_) #? 要不要G_.detach()? 这里可以加，因为算的是D的loss
D_fake_loss = self.BCE_loss(D_fake, self.y_fake_)

# 4. Summation the loss to get D_loss
D_loss = D_real_loss + D_fake_loss

self.train_hist['D_loss'].append(D_loss.item()) # Do not delete this
line

# 5. Use D_loss to update discriminator
D_loss.backward()
self.D_optimizer.step()

# 6. prepare G network
self.G_optimizer.zero_grad()

# 7. Do 3 again, this time compute loss with self.y_real_ as G_loss
```

```
G_ = self.G(z_)
D_fake = self.D(G_) #? 要不要detach? 这里不能加，因为会影响G
G_loss = self.BCE_loss(D_fake, self.y_real_)

self.train_hist['G_loss'].append(G_loss.item())

# 8. Use G_loss to update the generator
G_loss.backward()
self.G_optimizer.step()
```

Here is part of the printed results:

```
Epoch: [48] [ 300/ 937] D_loss: 0.99845302, G_loss: 2.29226398
Epoch: [48] [ 400/ 937] D_loss: 0.23266637, G_loss: 3.69020629
Epoch: [48] [ 500/ 937] D_loss: 0.17261446, G_loss: 4.11672306
Epoch: [48] [ 600/ 937] D_loss: 0.57366723, G_loss: 3.69269347
Epoch: [48] [ 700/ 937] D_loss: 0.17914708, G_loss: 3.35861802
Epoch: [48] [ 800/ 937] D_loss: 0.20649503, G_loss: 3.24884462
Epoch: [48] [ 900/ 937] D_loss: 0.12280109, G_loss: 3.97311926
Epoch: [49] [ 100/ 937] D_loss: 0.16323878, G_loss: 3.77830744
Epoch: [49] [ 200/ 937] D_loss: 0.21788222, G_loss: 2.92248821
Epoch: [49] [ 300/ 937] D_loss: 0.23028524, G_loss: 3.35537004
Epoch: [49] [ 400/ 937] D_loss: 0.21191353, G_loss: 4.25230503
Epoch: [49] [ 500/ 937] D_loss: 0.36970383, G_loss: 4.65532970
Epoch: [49] [ 600/ 937] D_loss: 0.34366137, G_loss: 5.42168999
Epoch: [49] [ 700/ 937] D_loss: 0.48647162, G_loss: 3.41435814
Epoch: [49] [ 800/ 937] D_loss: 0.27828896, G_loss: 4.81245804
Epoch: [49] [ 900/ 937] D_loss: 0.25245830, G_loss: 4.34694386
Epoch: [50] [ 100/ 937] D_loss: 0.12250993, G_loss: 4.78464699
Epoch: [50] [ 200/ 937] D_loss: 0.46119922, G_loss: 3.53125334
Epoch: [50] [ 300/ 937] D_loss: 0.29351741, G_loss: 4.07802677
Epoch: [50] [ 400/ 937] D_loss: 0.28858879, G_loss: 2.72849774
Epoch: [50] [ 500/ 937] D_loss: 0.17629588, G_loss: 3.92651987
Epoch: [50] [ 600/ 937] D_loss: 0.20826811, G_loss: 2.50049853
Epoch: [50] [ 700/ 937] D_loss: 0.39011145, G_loss: 3.56785583
Epoch: [50] [ 800/ 937] D_loss: 0.18085563, G_loss: 3.05343747
Epoch: [50] [ 900/ 937] D_loss: 0.13149780, G_loss: 5.03094101
Avg one epoch time: 18.77, total 50 epochs time: 938.71
Training finish!... save training results
 [*] Training finished!
 [*] Testing finished!
```

Now let's see the generated images in `./results/mnist/GAN/` ! Because of the space limit, we only compare the 1, 25,50 image here. The complete generated images are contained in the folder.

Fig1. GAN_epoch001

Fig2. GAN_epoch050

We can see that in the end, the picture generated by `G` is easy to be recognized and the picture can hardly be distincted from true ones! So our training process is valid and successful.