به نام خدا

دانشگاه تهران

دانشکده مهندسی برق و کامپیوتر

مدل‌های مولد عمیق

تمرین شماره2

نام و نام خانوادگی	مهسا ندافی قهنویه
شماره دانشجویی	۸۱۰۱۰۰۴۹۰
تاریخ ارسال گزارش	۱۴۰۲/۹/۵

فهرست گزارش سوالات

سوال ۱

الف)

همانطور که در صورت سوال ذکر شده است برای p و q داریم:

$$q(z \mid x) = N\big(2; \mu(x), \operatorname{diag}\left(\sigma^2(x)\right)\big)$$
$$P(z) = N(z; 0, z)$$
$$\mid \operatorname{diag}\left(\sigma^2(x)\right)\mid = \prod_{i=1}^{k} \sigma_i^2(x)$$
$$D_{KL}(q(z \mid x) \parallel p(z)) = \int q(z \mid x) \log\left(\frac{q(z \mid x)}{p(z)}\right) dz$$
$$= E_{q(z|x)}[\log q(z \mid x) - \log p(z)]$$
$$\log\ q(z \mid x) = -\frac{1}{2}(k\log(2\pi) + \log|\operatorname{diag}(\sigma^2(x))|$$
$$+(z-\mu(x))^{\top}\left(\operatorname{diag}\left(\sigma^{-2}(x)\right)\right)(z-\mu(x))$$

$$\log p(z) = -\frac{1}{2}(\mathrm{klog}(2\pi) + \log|I| + z^T I^{-1} z)$$

$$\log\ p(z) = -\frac{1}{2}(k\log(2\pi) + z^T z)$$

$$E_{q(z|x)}[\log q(z \mid x) - \log p(z)] = E_{q(z|x)}[-\tfrac{1}{2}\big(\mathrm{klog}(2\pi) +$$
$$\left(z-\mu(x)\right)^{\top}\big(\operatorname{diag}\big(\sigma^{-2}(x)\big)\big)\big(z-\mu(x)\big) + \log\big|\operatorname{diag}\big(\sigma^2(x)\big)\big| +$$
$$\tfrac{1}{2}(\mathrm{klog}(2\pi) + z^T z)\Big]$$
$$= \frac{1}{2}\big[tr\big(\operatorname{diag}\big(\sigma^2(x)\big)\big) + \mu(x)^T\mu(x) - 1 - \log\big|diag\big(o^2(x)\big)\big|\big]$$
$$= \frac{1}{2}\sum_{i=1}^{k}\left(\sigma_i^2(x) + \mu_i^2(x) - 1 - \log\ \sigma_i^2(x)\right)$$

ب)

$$\log \mathrm{P}_\theta(x) = \log \mathbb{E}_z[f(x,z) = \frac{p_\theta(x,z)}{q_\varphi(z \mid x)}] \overset{\text{Jensen}}{\geq} \mathbb{E}_z[f(x,z) = \frac{p_\theta(x,z)}{q_\varphi(z \mid x)}]$$

$$f(x, z_1, \dots, z_m) = \frac{1}{m} \sum_{i=1}^{m} \frac{p_\theta(x, z_i)}{q_\varphi(z_i \mid x)}$$

$$\log \left(\mathbb{E}_z \left(\frac{p_\theta(x,z)}{q_\varphi(z \mid x)} \right) \right) \cong \frac{1}{m} \sum_{i=1}^{m} \mathbb{E}_{q(z_0,\dots,z_k)} [\frac{p_\theta(x, z_i)}{q_\varphi(z_i \mid x)}]$$

$$= \frac{1}{m} \sum_{i=1}^{m} \mathbb{E}_z \left[\frac{p_\theta(z_i|x)p(x)}{q_\varphi(z_i \mid \mathrm{x})} \right] = \sum_i \mathbb{E}_z[p_\theta(x)] = \log (p_\theta(x))$$

$$\log (p_\theta(x)) \overset{\text{Jensen}}{\geq} \mathbb{E}_{z_0,\dots,z_m \sim q_\varphi(\mathrm{z}|x)} [log \frac{1}{m} \sum_{i=1}^{m} \frac{p_\theta(x, z_i)}{q_\varphi(z_i \mid x)}]$$

در نتیجه $\mathbb{E}_{z_0,\dots,z_m \sim q_\varphi(\mathrm{z}|x)} [log \frac{1}{m} \sum_{i=1}^{m} \frac{p_\theta(x,z_i)}{q_\varphi(z_i|x)}]$ را به جای likelihoodماکزیمم میکنیم.

ج)

در واقع چالش posterior collapse در VAE به این معنی می باشد که توزیع variational یادگرفته شده $q_\varphi(z \mid x)$ بسیار به توزیع prior ($p(z)$) نزدیک می شود. به عبارت دیگر در طول یادگیری، ترم $D_{KL}\left(q_\varphi\left(z \mid x\right) \| \, p(z)\right)$ در تابع loss مان به صفر می رسد مع این امر باعث می شود که توزیع posterior مان برابر توزیع prior مان شود که در این صورت $q_\varphi(z \mid x)$ مستقل از مقدار x (ورودی مان) شده و هیچ information ای از ورودی هایمان را حمل نخواهد کرد. برای حل این مشکل میتوان معماری مدل را برای متعادل کردن قدرت encoder , decoder تغییر داد همچنین میتوان برای حفظ اطلاعات از skip connection ها بین encoder,decoder استفاده کرد. اگر این مسئله مربوط به داده های ما باشد میتوان از data augmentation برای حل آن استفاده کرد.

یک راه حل می تواند این باشد که ترم $D_{KL}\left(q_\varphi\left(z\,|x\right)\|\,p(z)\right)$ را طوری تنظیم کنیم که مقدارش هیچ وقت به صفر نرسد. مثلا می توان یک constraint به صورت زیر روی این ترم اعمال کرد:

$$D_{KL}\left(q_\varphi\left(z\,|x\right)\|\,p(z)\right)\geq\delta\quad;\quad\delta>0$$

با این کار می توان تضمین کرد که $q_\varphi(z\,|x)$ هیچ وقت برابر p(z) نشده و با تنظیم δ نیز می توان میزان نزدیکی این دو توزیع prior و posterior را کنترل کرد طوری که متناسب با کاربردمان بتوانیم تضمین کنیم که posterior collapse نخواهیم داشت.

د)

$$\log\left(p_\theta(x)\right)\geq\mathbb{E}_{q_\varphi(z|x)}[logp_\theta(x|z)]-\mathrm{KL}(q_\varphi(z\mid x)||p(z))$$

$$\log\,p_\theta(x)=\log\int_z p_\theta(x,z)dz$$

$$=\quad\log\int_z p_\theta(x,z)\frac{q_\phi(z\mid x)}{q_\phi(z\mid x)}dz\geqslant$$

$$\int q_\phi(z\mid x)\log\frac{p_\theta(x,z)}{q_\phi(z\mid x)}dz=$$

$$E_{q_\phi(z|x)}\left[\log\frac{p_\theta(x\mid z)\cdot p_\theta(z)}{q_\phi(z\mid x)}\right]$$

$$=\quad E_{q_\phi(z|x)}[\log\,p_\theta(x\mid z)]+E_{q_\phi(z|x)}\left[\log\frac{p_\theta(z)}{q_\phi(z\mid x)}\right]$$

$$=\quad E_{q_\phi(z|x)}[\log\,p_\theta(x\mid z)]+KL(q_\phi(z\mid x)\,\|\,p_\theta(z))$$

$$\Rightarrow\quad\log\,P_\theta(x)\geqslant ELBO(x;\theta,\phi)$$

باید توجه شود که در عبارت بالا عبارت KL همیشه مقداری بزرگتر از صفر دارد پس میتوان برای ماکزیمم کردن likelihood از ELBO استفاده کرد.

سوال ۲ – VAE

الف) همان طور که خواسته شده است کد مربوط به Reparameterization Trick در تابع sample_gaussian وارد شده است. شکل زیر قطعه کد مربوطه را نشان میدهد همان طور که مشاهده میشود این تابع میانگین و واریانس توزیع گوسی $q_\phi(z|x)$ را میگیرد و یک نمونه از آن را در خروجی برمیگرداند:

```
def sample_gaussian(m, v):
    """
    Element-wise application reparameterization trick to sample from
Gaussian

    Args:
        m: tensor: (batch, ...): Mean
        v: tensor: (batch, ...): Variance

    Return:
        z: tensor: (batch, ...): Samples
    """

################################################################################
#####
    # TODO: complete the code here
    # Task: Sample z
    N_dist=torch.distributions.Normal(0, 1)
    N_dist.loc = N_dist.loc.cuda()
    N_dist.scale = N_dist.scale.cuda()
    epsilon = N_dist.sample(m.shape)
    z = m + torch.sqrt(v)*epsilon

    # End

################################################################################
#####

    return z
```

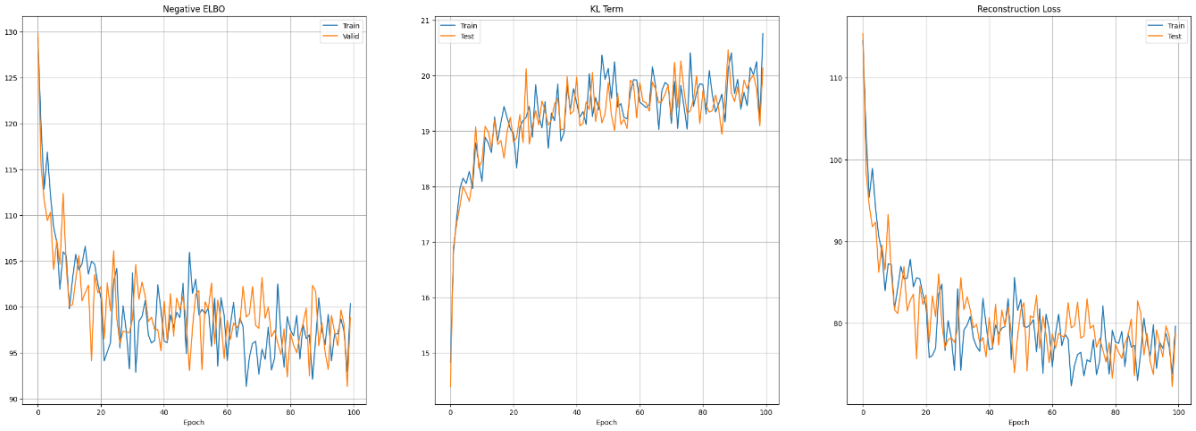
ب)

کد این بخش در تابع NELBO به شکل زیر پیاده شده است:

```
def negative_elbo_bound(self, x):
        """
        Computes the Evidence Lower Bound, KL and, Reconstruction costs

        Args:
```

```
            x: tensor: (batch, dim): Observations

        Returns:
            nelbo: tensor: (): Negative evidence lower bound
            kl: tensor: (): ELBO KL divergence to prior
            rec: tensor: (): ELBO Reconstruction term
        """

###############################################################################
#####
        # TODO: complete the code here
        # Task: Compute negative Evidence Lower Bound and its KL and
Reconstruction term
        # Note that nelbo = kl + rec
        # Outputs should all be scalar
        qm, qv = self.enc.encode(x)
        kl=
kl_normal(qm,qv,pm=torch.zeros(1).expand(self.z_dim).cuda(),pv=torch.ones(1
).expand(self.z_dim).cuda())
        kl=torch.mean(kl)
        z = sample_gaussian(qm, qv)
        logits= self.dec.decode(z)
        recs= -log_bernoulli_with_logits(x, logits)
        recs=torch.mean(recs)
        nelbo= kl+recs

        # End

###############################################################################
#####

        return nelbo, kl, recs
```

ج) تست و ترین بر روی مدل در ۱۰۰ ایپاک صورت پذیرفته و در شکل ۱ تابع خطای مربوط به ترین و تست آورده شده نمودار نارنجی مربوط به تست و نمودار آبی مربوط به ترین است.



شکل۱: نمودار تغییرات **Negative ELBO**، **KL term** و **Reconstruction term** در طول زمان یادگیری برای داده های ترین و تست

همانطور که انتظار میرود نمودار kl بصورت صعودی است چرا که شبکه در تلاش است این مقدار را max کند و مقدار ELBO- را مینیمم کند تا برابر با احتمال Evidence که مقدار ثابتی است شود.

کد مربوط به این بخش در زیر آورده شده است:

```
vae = VAE(z_dim=z).to(device)
optimizer = optim.Adam(vae.parameters(), lr=learning_rate)
nelbo=torch.zeros(1,100)
kl=torch.zeros(1,100)
rec=torch.zeros(1,100)
nelbo_test=torch.zeros(1,100)
kl_test=torch.zeros(1,100)
rec_test=torch.zeros(1,100)

for i in tqdm(range(iter_max)):
    for batch_idx, (xu, yu) in enumerate(train_loader):

        optimizer.zero_grad()

        xu = torch.bernoulli(xu.to(device).reshape(xu.size(0), -1))
        yu = yu.new(np.eye(10)[yu]).to(device).float()
        loss, summaries = vae.loss(xu)
        nelbo[0][i]=loss
        kl[0][i]=summaries['gen/kl_z']
        rec[0][i]=summaries['gen/rec']
        loss.backward()
        optimizer.step()

    print('nelbo: ', summaries['train/loss'],'kl:
',summaries['gen/kl_z'],'rec: ',summaries['gen/rec'])
    with torch.no_grad():
      for batch_idx_test, (xu_test, yu_test) in enumerate(test_loader):
```

```
        optimizer.zero_grad()

        xu_test =
torch.bernoulli(xu_test.to(device).reshape(xu_test.size(0), -1))
        yu_test = yu_test.new(np.eye(10)[yu_test]).to(device).float()
        loss_test, summaries_test = vae.loss(xu_test)
        nelbo_test[0][i]=loss_test
        kl_test[0][i]=summaries_test['gen/kl_z']
        rec_test[0][i]=summaries_test['gen/rec']


################################################################################
#####
    # TODO: complete the code here
    # report the loss terms across time for train and validation datasets
    #
    # You can choose to only perform validation once in every 10 (Or any
other number
    # depending on the speed of the algorithm on your machine) iterations
and display
    # the training and validation statistics on those iterataions.

    # End

################################################################################
#####
```

خروجی به هنگام آموزش:

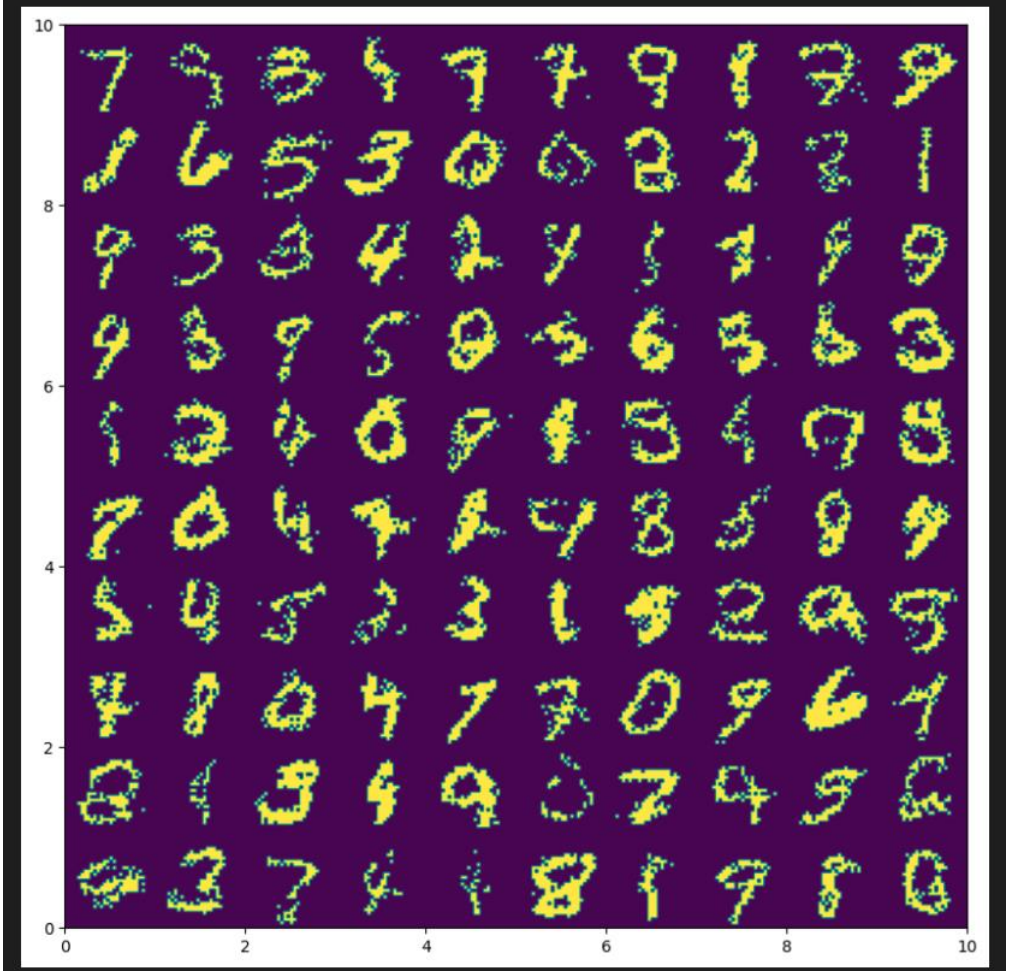
```
95%|██████████ | 95/100 [15:45<00:48, 9.78s/it]
nelbo: tensor(97.0606, device='cuda:0', grad_fn=<AddBackward0>) kl: tensor(20.1531,
device='cuda:0', grad_fn=<MeanBackward0>) rec: tensor(76.9075, device='cuda:0',
grad_fn=<MeanBackward0>)
96%|██████████ | 96/100 [15:56<00:39, 9.89s/it]
nelbo: tensor(98.7200, device='cuda:0', grad_fn=<AddBackward0>) kl: tensor(20.0163,
device='cuda:0', grad_fn=<MeanBackward0>) rec: tensor(78.7038, device='cuda:0',
grad_fn=<MeanBackward0>)
97%|██████████| 97/100 [16:06<00:29, 9.95s/it]
nelbo: tensor(97.3411, device='cuda:0', grad_fn=<AddBackward0>) kl: tensor(20.2528,
device='cuda:0', grad_fn=<MeanBackward0>) rec: tensor(77.0883, device='cuda:0',
grad_fn=<MeanBackward0>)
98%|██████████| 98/100 [16:16<00:19, 9.94s/it]
nelbo: tensor(92.9630, device='cuda:0', grad_fn=<AddBackward0>) kl: tensor(19.1649,
device='cuda:0', grad_fn=<MeanBackward0>) rec: tensor(73.7980, device='cuda:0',
grad_fn=<MeanBackward0>)
99%|██████████| 99/100 [40:24<07:21, 441.39s/it]
nelbo: tensor(100.3641, device='cuda:0', grad_fn=<AddBackward0>) kl: tensor(20.7600,
device='cuda:0', grad_fn=<MeanBackward0>) rec: tensor(79.6041, device='cuda:0',
grad_fn=<MeanBackward0>)
```

100%|██████████| 100/100 [40:34<00:00, 24.35s/it]

۱۰۰ تصویر با استفاده از کد زیر تولید شده و در شکل ۲ نمایش داده شده است.

```
def plot_reconstructed(batch_img, r0=(-5, 5), r1=(-5, 5), n=10):
    w = 28
    img = np.zeros((n*w, n*w))
    for i, y in enumerate(np.linspace(*r1, n)):
        for j, x in enumerate(np.linspace(*r0, n)):
            img[(n-1-i)*w:(n-1-i+1)*w, j*w:(j+1)*w] = batch_img[i*10+j]
    plt.imshow(img, extent=[*r0, *r1])

plt.figure(figsize=(10, 10))
img_100=vae.sample_x(100)
print(img_100.shape)
img_100=img_100.reshape(100,28, 28).to('cpu').detach().numpy()
plot_reconstructed(img_100, r0=(0, 10), r1=(0, 10))نمو
```

```
torch.Size([100, 784])
```



شکل۲: صد تصویر تولید شده به وسیله شبکه بعد از ۱۰۰ ایپاک آموزش

این تصاویر نشان دهنده اعداد تولید شده توسط VAE پس از یادگیری توزیع دادههای MNIST هستند. این تصاویر نشان میدهند که مدل میتواند دادههای جدیدی را ایجاد کند که شبیه به داده های آموزشی است، که نشان میدهد یک مدل خوب یاد گرفته شده است.

د)کد مربوط به IWAE در تابع مربوط به آن به شکل زیر نوشته شده است:

$$\text{IWAE} = \mathbb{E}_{z\sim q_\phi(z|x)}\left[\frac{1}{K}\sum_{k=1}^{K}\left(\log p_\theta(x|z^{(k)}) + \log p(z^{(k)}) - \log q_\phi(z^{(k)}|x)\right)\right]$$

```
def negative_iwae_bound(self, x, iw):
        """
        Computes the Importance Weighted Autoencoder Bound
        Additionally, we also compute the ELBO KL and reconstruction terms

        Args:
            x: tensor: (batch, dim): Observations
            iw: int: (): Number of importance weighted samples

        Returns:
            niwae: tensor: (): Negative IWAE bound
            kl: tensor: (): ELBO KL divergence to prior
            rec: tensor: (): ELBO Reconstruction term
        """

################################################################################
#####
        # TODO: Modify/complete the code here
        # Compute niwae (negative IWAE) with iw importance samples, and the
KL
        # and Rec decomposition of the Evidence Lower Bound
        #
        # Outputs should all be scalar

################################################################################
#####
        batch = x.shape[0]
        multi_x = duplicate(x, iw)

        qm, qv = self.enc.encode(x)
        multi_qm =duplicate(qm, iw)
        multi_qv = duplicate(qv, iw)
```

```
        # z will be (batch*iw x z_dim)
        # with sampled z's for a given x non-contiguous!
        z =sample_gaussian(multi_qm,multi_qv)

        logits = self.dec.decode(z)
        recs = log_bernoulli_with_logits(multi_x, logits)

        multi_pm = self.z_prior[0].expand(multi_qm.shape)
        multi_pv = self.z_prior[1].expand(multi_qv.shape)

        pm, pv = self.z_prior
        log_qz = log_normal(z, multi_qm, multi_qv)
        log_pz = log_normal(z, pm.expand_as(multi_qm),
pv.expand_as(multi_qv))

        unflat_log_ratios = recs + log_pz - log_qz

        niwae = -log_mean_exp(unflat_log_ratios.reshape(iw, -1),
dim=0).mean()
        niwae = niwae.mean()
        kl = kl_normal(multi_qm, multi_qv, pm.expand_as(multi_qm),
pv.expand_as(multi_qv)).mean()
        rec = recs.mean()

##########################################################################
#####
        # End of code modification

##########################################################################
#####
        return niwae, kl, rec
```

ه) به منظور پیاده سازی در m های خواسته شده کد زیر نوشته شده است:

```
vae = VAE(z_dim=z).to(device)
optimizer = optim.Adam(vae.parameters(), lr=learning_rate)

m = [5, 50, 150]
repetition = 50

with torch.no_grad():
    test_elbo_loss = 0.0
    for x, _ in test_loader:
        x = x.to(device).reshape(x.size(0), -1)
        elbo_neg, _, _ = vae.negative_elbo_bound(x)
        test_elbo_loss += elbo_neg.item()
    mean_elbo_neg = test_elbo_loss / len(test_loader)
    print(f"Negative ELBO: {mean_elbo_neg}")
```

```
for each_m in m:
    total_niwae = 0
    for _ in tqdm(range(repetition), desc=f"Repetition in m={each_m}"):
        test_running_niwae = 0.0
        for x, _ in test_loader:
            x = torch.bernoulli(x.to(device).reshape(x.size(0), -1))
            niwae, _, _ = vae.negative_iwae_bound(x, iw=each_m)
            test_running_niwae += niwae.item()

        mean_niwae = test_running_niwae / len(test_loader)
        total_niwae += mean_niwae

    final_mean_niwae = total_niwae / repetition
    print(f"Negative IWAE in m={each_m}: {final_mean_niwae}")
```

خروجی:

```
Negative ELBO: 545.8072595214844
Repetition in m=5: 100%|██████████| 50/50 [01:13<00:00, 1.46s/it]
Negative IWAE in m=5: 544.9184030883789
Repetition in m=50: 100%|██████████| 50/50 [01:22<00:00, 1.64s/it]
Negative IWAE in m=50: 544.5022906127929
Repetition in m=150: 100%|██████████| 50/50 [01:47<00:00, 2.16s/it]
Negative IWAE in m=150: 544.4429091308593
```

برای مجموعه داده آزمایشی، میانگین IWAE بیش از ۵۰ تکرار برای مقادیر مختلف m محاسبه شده است.

این نتایج نشان میدهد که با افزایش تعداد نمونه های وزندار اهمیت، میانگین کران IWAE منفی اندکی کاهش مییابد، که نشان دهنده کران پایینتر بهتر است. این روند با انتظارات IWAE مطابقت دارد.

سوال ۳ – Normalizing flow

الف)

$$p(z) = N(N \mid 0,1) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}z^2}$$

$$x = 1.5z + 3 \Rightarrow z = \frac{x-3}{1.5} = \frac{x}{1.5} - 2$$

$$p(x) = p(z = f^{-1}(x)) \left| \det \left(\frac{\partial f^{-1}(x)}{x} \right) \right|$$

$$p(x) = \frac{2}{3} \times P_z(z = f^{-1}(x))$$

$$p(x) = \frac{2}{3} \times \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x}{1.5}-2\right)^2}$$

$$p(x) = \frac{\sqrt{2}}{3\sqrt{\pi}} e^{-\frac{1}{2}\left(\frac{x}{1.5}-\frac{3}{1.5}\right)^2} = \frac{1}{1.5\sqrt{2\pi}} e^{-\frac{1}{2}\cdot\left(\frac{x-3}{1.5}\right)^2}$$

$$p(x) = N\ 3\ 1.5^2$$

ب)

$$y_a = x_a,$$
$$y_b = \exp\left(s(x_a)\right) \otimes x_b + t(x_a)$$

$$\mathbf{J} = \begin{bmatrix} \frac{\partial \mathbf{f}}{\partial x_1} & \cdots & \frac{\partial \mathbf{f}}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

$$\mathbf{J} = \begin{pmatrix} \mathbf{I} & 0 \\ \frac{\partial y_b}{\partial x_a} & \operatorname{diag}\left(\exp\left(s(x_a)\right)\right) \end{pmatrix}$$

برای محاسبه دترمینان یک ماتریس بالا مثلثی کافی است ضرایب قطر اصلی را در یکدیگر ضرب کنیم:

$$\det(\mathbf{J}) = \det(\mathbf{I}) \times \det\left(\operatorname{diag}\left(\exp\left(s(x_a)\right)\right)\right)$$

$$\det(\mathbf{J}) = 1 \times \prod_i \exp\left(s_i(x_a)\right) = \exp\left(\sum_i s_i(x_a)\right)$$

$$|\det(\mathbf{J})| = \exp\left(\sum_i s_i(x_a)\right)$$

د) به منظور پیاده سازی real nvp ابتدا داده های mnist را فراخوانی کرده:

```
# MNIST dataset
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Lambda(lambda x: x.view(-1))
])

train_dataset = datasets.MNIST(root='./data', train=True,
transform=transform, download=True)
test_dataset = datasets.MNIST(root='./data', train=False,
transform=transform)
num_epochs = 60
batch_size = 100
train_loader = DataLoader(dataset=train_dataset, batch_size=batch_size,
shuffle=True)
test_loader = DataLoader(dataset=test_dataset, batch_size=batch_size,
shuffle=False)
```

سپس لایه کوپلینگ طبق فرمول بخش ب یه شکل زیر پیاده سازی شد:

```
class CouplingLayer(nn.Module):
    def __init__(self, in_channels, hidden_channels):
        super(CouplingLayer, self).__init__()

        self.s_net = nn.Sequential(
            nn.Linear(in_channels // 2, hidden_channels),
            nn.BatchNorm1d(hidden_channels),
            nn.ReLU(),
```

```
            nn.Linear(hidden_channels, hidden_channels//2),
            nn.BatchNorm1d(hidden_channels//2),
            nn.ReLU(),
            nn.Linear(hidden_channels//2, in_channels // 2)
        )
        self.t_net = nn.Sequential(
            nn.Linear(in_channels // 2, hidden_channels),
            nn.BatchNorm1d(hidden_channels),
            nn.ReLU(),
            nn.Linear(hidden_channels, hidden_channels // 2),
            nn.BatchNorm1d(hidden_channels // 2),
            nn.ReLU(),
            nn.Linear(hidden_channels // 2, in_channels // 2)
        )

    def forward(self, x):
        x_a, x_b = x.chunk(2, dim=1)
        s = self.s_net(x_a)
        t = self.t_net(x_a)
        t = t.to(device)
        s = s.to(device)
        y_b = x_b * torch.exp(s) + t
        y = torch.cat([x_a, y_b], dim=1)
        return y, torch.sum(s, dim=1)

    def inverse(self, y):
        y_a, y_b = y.chunk(2, dim=1)
        s = self.s_net(y_a)
        t = self.t_net(y_a)
        t = t.to(device)
        s = s.to(device)
        x_b = (y_b - t) * torch.exp(-s)
        x = torch.cat([y_a, x_b], dim=1)
        return x
```

معماری شبکهای s,t یک مدل سری خطی ۳ لایه است که در کد بالا دیده میشود.

سپس مدل اصلی به شکل زیر پیاده شده:

```
class RealNVP(nn.Module):
    def __init__(self, num_coupling_layers, in_channels, hidden_channels,
input_size):
        super(RealNVP, self).__init__()

        self.num_coupling_layers = num_coupling_layers
        self.coupling_layers = nn.ModuleList([
            CouplingLayer(in_channels, hidden_channels) for _ in
range(num_coupling_layers)
```

```
        ])
        self.distribution =
MultivariateNormal(torch.zeros(input_size).to(device),
torch.eye(input_size).to(device))

    def forward(self, x):
        for i in range(self.num_coupling_layers):
            x, _ = self.coupling_layers[i](x)
            x = self.permutation(x)
        return x

    def inverse(self, y):
        for i in reversed(range(self.num_coupling_layers)):
            y = self.coupling_layers[i].inverse(y)
            y = self.permutation(y)
        return y

    def jacobian_det(self, x):
        log_det_J = 0.0
        for i in range(self.num_coupling_layers):
            y, jacobian_det = self.coupling_layers[i](x)
            log_det_J += jacobian_det
            x = self.permutation(y)
        final_loss = log_det_J + self.distribution.log_prob(x)
        return final_loss

    def permutation(self, x):
        return torch.flip(x, dims=[1])
```

توجه شود بعد از هر لایه کوپلینگ permutation نیز اجرا میشود و این امر باید در محاسبه loss نیز تاثیر داده شود تا شبکه به خوبی آموزش داده شود برای تعریف loss که همان متد jacobian_det در کلاس مدل RealNVP است از فرمول زیر استفاده شده است:

$$\max_{\theta} \log p_X(\mathcal{D};\theta) = \sum_{x\in\mathcal{D}} \log p_Z\left(f_\theta^{-1}(x)\right) + \log\left|\det\left(\frac{\partial f_\theta^{-1}(x)}{\partial x}\right)\right|$$

که بصورت زیر پیاده شده است:

```
final_loss = log_det_J + self.distribution.log_prob(x)
```

برای ترین و تست ۲ تابع زیر نوشته شده اند:

```
# Training function
def train_realnvp(model, train_loader, optimizer, num_epochs):
    losses = []
```

```
    for epoch in range(num_epochs):
        model.train()
        total_loss = 0.0

        for batch in train_loader:
            data = batch[0].to(device)
            optimizer.zero_grad()
            output = model(data)
            loss = -model.jacobian_det(data).mean()  # Negative log-
likelihood
            loss.backward()
            optimizer.step()
            total_loss += loss.item()

        average_loss = total_loss / len(train_loader)
        print(f'Epoch [{epoch + 1}/{num_epochs}], Loss:
{average_loss:.1f}')
        losses.append(average_loss)

    return losses


# Testing function
def test_realnvp(model, test_loader):
    model.eval()
    total_loss = 0.0
    losses = []

    with torch.no_grad():
        for batch in test_loader:
            data = batch[0].to(device)
            output = model(data)
            try:
                loss = -model.jacobian_det(data).mean()  # Negative log-
likelihood
                total_loss += loss.item()
                losses.append(total_loss)
            except ValueError as e:
                continue

    if len(losses) > 0:
        average_loss = total_loss / len(losses)
        print(f'Test Loss: {average_loss:.1f}')
    else:
        print("No valid losses computed.")

    return losses
```

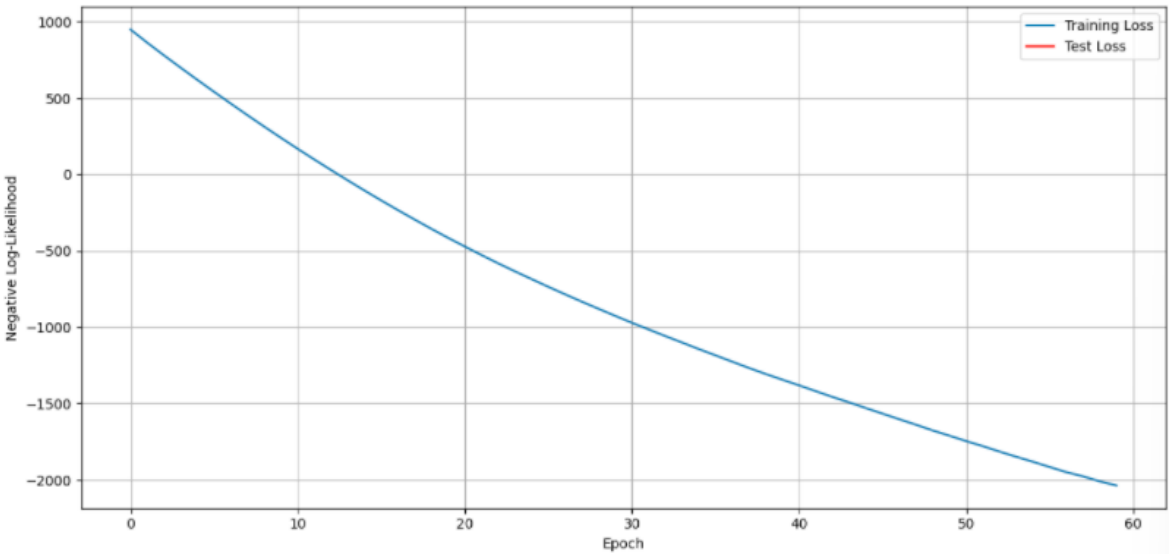
در نهایت مدل ساخته شده و آموزش داده شد:

```
# Model configuration
num_coupling_layers = 4
in_channels = 784  # Assuming flattened MNIST images of size 28x28
hidden_channels = 512
input_size = in_channels
realnvp_model = RealNVP(num_coupling_layers, in_channels, hidden_channels,
input_size).to(device)

# Loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(realnvp_model.parameters(), lr=1e-6)
# Training the RealNVP model
train_losses = train_realnvp(realnvp_model, train_loader, optimizer,
num_epochs=num_epochs)
checkpoint_path = f'realNVP_{num_epochs}epochs.pt'
torch.save(realnvp_model.state_dict(), checkpoint_path)
```

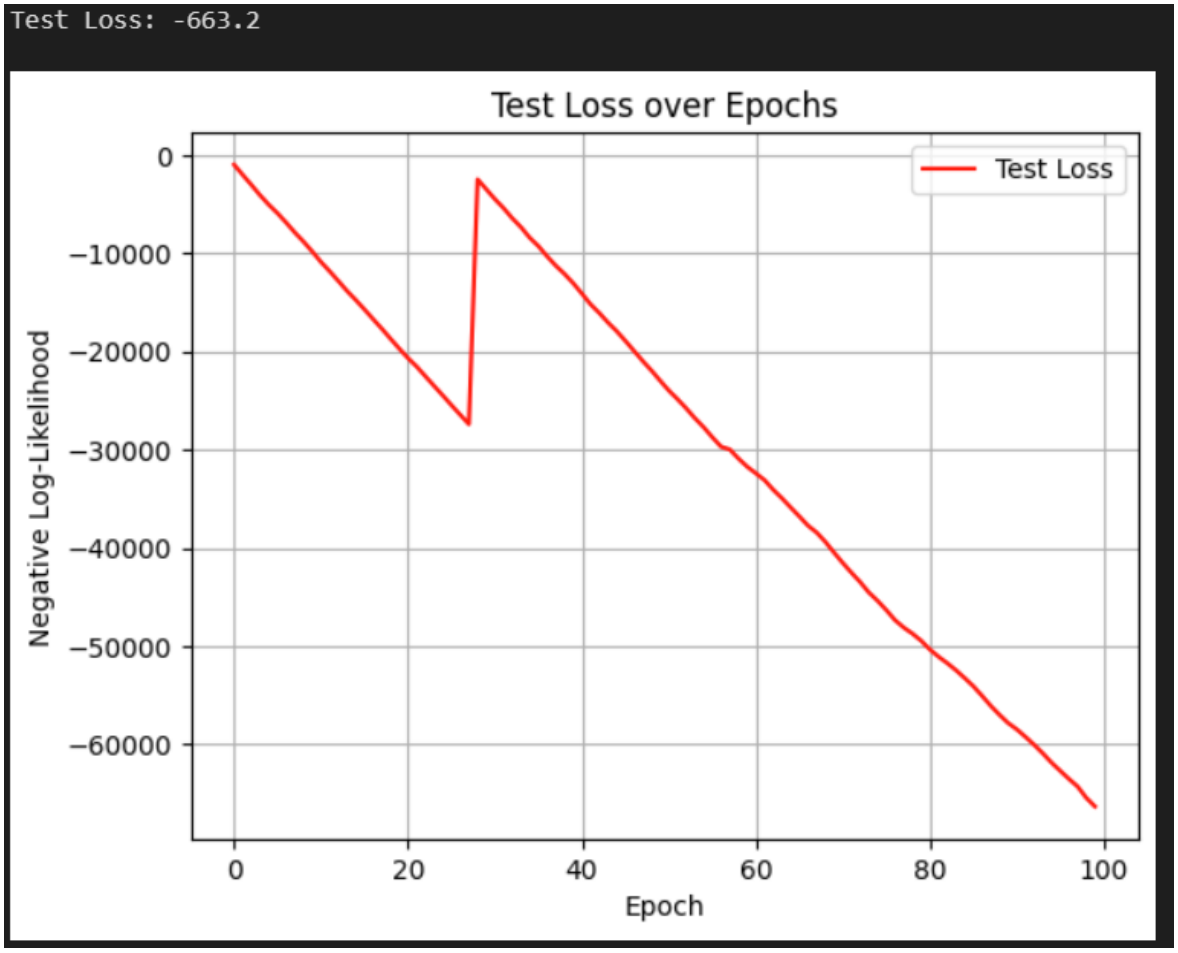
خروجی:

```
Epoch [24/60], Loss: -635.1 Epoch [25/60], Loss: -685.9
...
Epoch [57/60], Loss: -1949.6 Epoch [58/60], Loss: -1976.9 Epoch [59/60],
Loss: -2009.7 Epoch [60/60], Loss: -2037.0
```

توابع loss برای ترین و تست در شکل ۳و ۴ آورده شده است:



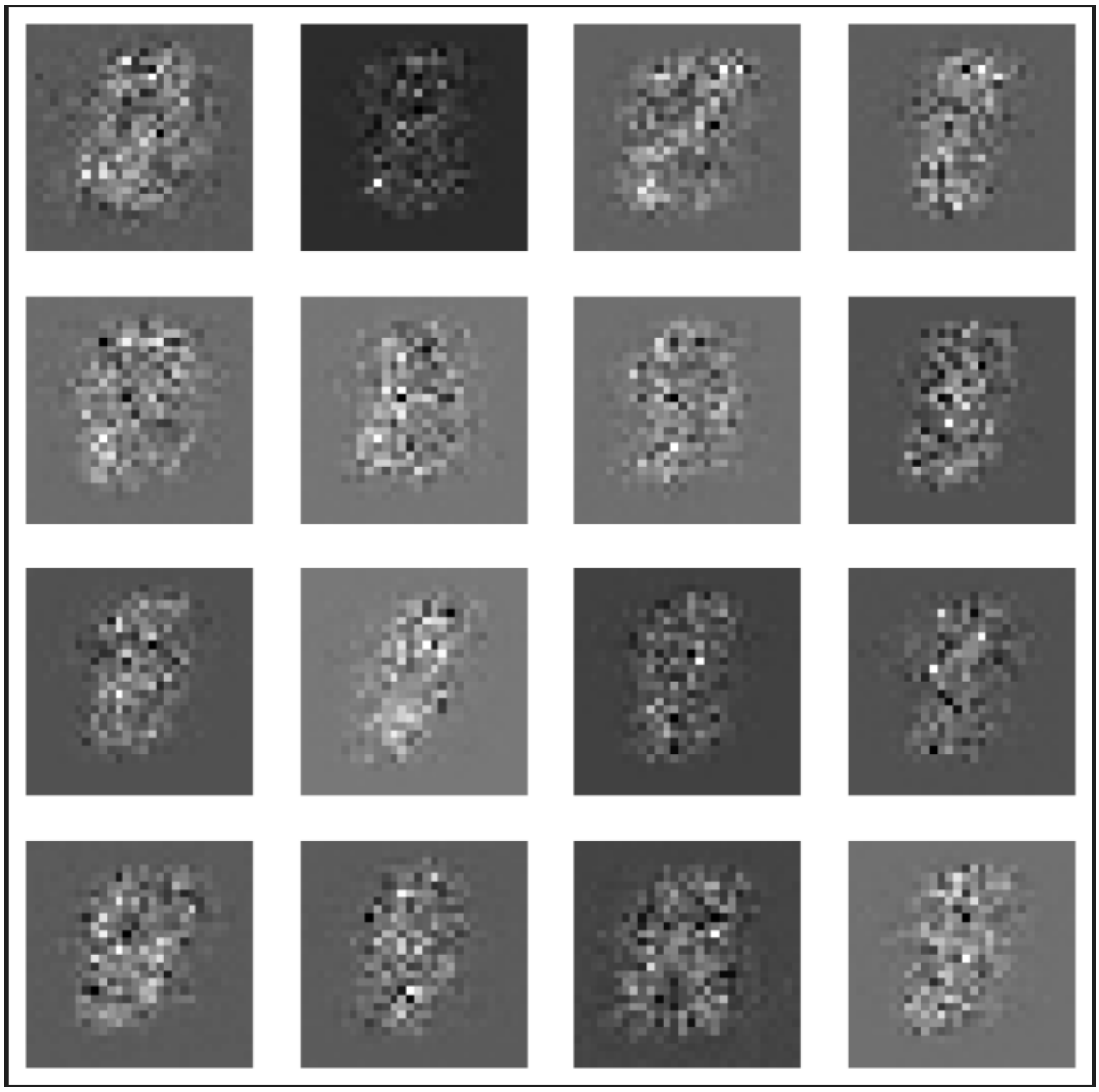
شکل۳: نمودار **loss** به هنگام ترین برای ۶۰ ایپاک



شکل۴:نمودار **loss** برای تست

همانطور که مشخص است نمودار های loss کاملا بصورت نزولی هستند که نشان دهنده آموزش درست مدل است.

در نهایت ۱۶ تصویر توسط شبکه آموزش داده شده از ۱۶ نمونه تصادفی تولید شد



شکل۵: ۱۶ تصویر تولید شده به وسیله شبکه بعد از ۶۰ ایپاک آموزش

همانطور که مشخص استخروجی از حالت نویز خارج شده و تقریبا میتوان عدد ۸ را در بین آنها دید.