پروژه درس شبکه عصبی

بخش اول

```
train_dataset = MNIST(root='./data', train=True, transform=img_transform, download=True)
train_dataloader = DataLoader(train_dataset, batch_size=128, shuffle=True)
test_dataset = MNIST(root='./data', train=False, transform=img_transform, download=True)
test_dataloader = DataLoader(test_dataset, batch_size=128, shuffle=False)
```

ابتدا دیتاست MNIST را دانلود می کنیم و در فولدر مربوطه ذخیره میکنیم. عکس ها به تنسور تبدیل می شوند و سپس یک dataloader می سازیم تا داده ها را در قالب batch هایی با اندازه ۱۲۸ نمونه بارگذاری می کند. دیتاست آموزشی به صورت تصادفی مرتب می شود ولی دیتاست تستی ثابت می ماند.

```
self.fc_mu = nn.Linear(in_features=c*2*7*7, out_features=latent_dims)
self.fc_logvar = nn.Linear(in_features=c*2*7*7, out_features=latent_dims)
```

```
x_mu = self.fc_mu(x)
x_logvar = self.fc_logvar(x)
```

این بخش درون کلاس انکودر است که پس از لایه های کانولوشنی دو لایه خطی ایجاد کردیم. ورودی های این لایه خروجی لایه قبلی است که c * 2 * 7 * 7 است و خروجی لایه متغیر latent_dims است که برابر 2 است.

fc_mu مقدار میانگین (μ) را برای متغیرهای نهفته تخمین می زند و fc_logvar مقدار $\log(\sigma^2)$ را تخمین می زند که برای نمونه گیری از فضای نهفته در VAE استفاده می شود.

```
self.conv2 = nn.ConvTranspose2d(in_channels=c*2, out_channels=c, kernel_size=4, stride=2, padding=1)
self.conv1 = nn.ConvTranspose2d(in_channels=c, out_channels=1, kernel_size=4, stride=2, padding=1)
```

```
x = self.conv2(x)
x = F.relu(x)
x = self.conv1(x)
```

این بخش درون کلاس دیکودر است که پس از یک لایه خطی 2 لایه کانولوشنی ایجاد کردیم که دقیقا بر عکس کار انکودر را می کند. 2 لایه کانولوشنی ایجاد کردیم که بین آن ها از ReLU استفاده کردیم.

برعکس Encoder که از Conv2d استفاده می کرد، اینجا از ConvTranspose2d برای افزایش ابعاد تصویر استفاده شده است.

```
# TODO: Defin an Encoder and a Decoder instans
self.encoder = Encoder()
self.decoder = Decoder()
```

در ادامه در کلاس VAE انکودر و دیکودری که ساختیم را تعریف می کنیم.

```
def vae_loss(recon_x, x, mu, logvar):

    recon_loss = F.binary_cross_entropy(recon_x.view(-1, 784), x.view(-1, 784), reduction='sum')
    kldivergence = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())

    return recon_loss + variational_beta * kldivergence
```

برای تعریف تابع vae_loss دو بخش اصلی داریم:

KL Divergence Loss + Reconstruction Loss

در قسمت اول از Binary Cross Entropy (BCE) Loss به عنوان استفاده شده است.
recon_x تصویر بازسازی شده و x تصویر اصلی است.
هدف این loss این است که می خواهیم تصویر بازسازی شده تا حد امکان به تصویر اصلی نزدیک باشد، یعنی مقدار BCE Loss کم شود.

در قسمت دوم فرمول KL Divergence را محاسبه می کند که اختلاف توزیع نهفته مدل و توزیع نرمال استاندارد را نشان می دهد.

$$D_{KL} = \frac{1}{2}\sum\left(1 + \log(\sigma^2) - \mu^2 - \sigma^2\right)$$

این واگرایی باعث می شود که توزیع نهفته تا حد ممکن شبیه یک توزیع نرمال استاندارد شود و در نتیجه مدل بتواند نمونه های جدید از این توزیع بگیرد.

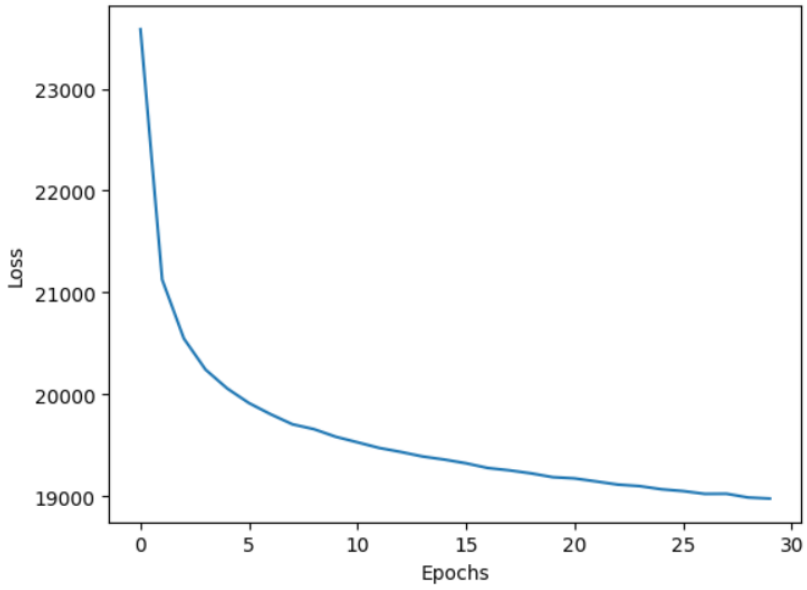
```
for image_batch, _ in train_dataloader:

    image_batch = image_batch.to(device)
    recon_images, mu, logvar = vae(image_batch)
    optimizer.zero_grad()
    loss = vae_loss(recon_images, image_batch, mu, logvar)
    loss.backward()
    optimizer.step()

    train_loss_avg[-1] += loss.item()
    num_batches += 1
```

این بخش برای آموزش مدل است که در هر ایپاک و برای هر بچ این کار ها صورت می گیرد:

انتقال داده به GPU در صورت وجود و حذف گرادیان های قبلی با optimizer.zero_grad()
با اجرای مدل vae(image_batch) خروجی بازسازی شده، میانگین و log واریانس را برمی گرداند.
سپس خطا محاسبه می شود و محاسبه گرادیان ها با loss.backward() صورت می گیرد و در نهایت به روزرسانی وزن های مدل انجام می شود.

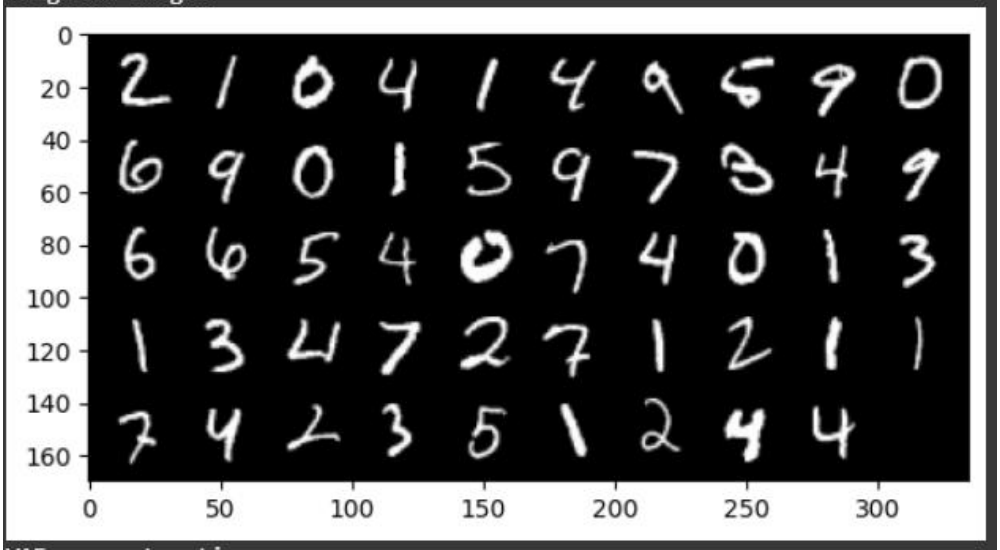
پس از فرایند آموزش به نتیجه زیر رسیدیم:

```
Epoch [28 / 30] average reconstruction error: 19024.928601
Epoch [29 / 30] average reconstruction error: 18987.903835
Epoch [30 / 30] average reconstruction error: 18977.575351
```

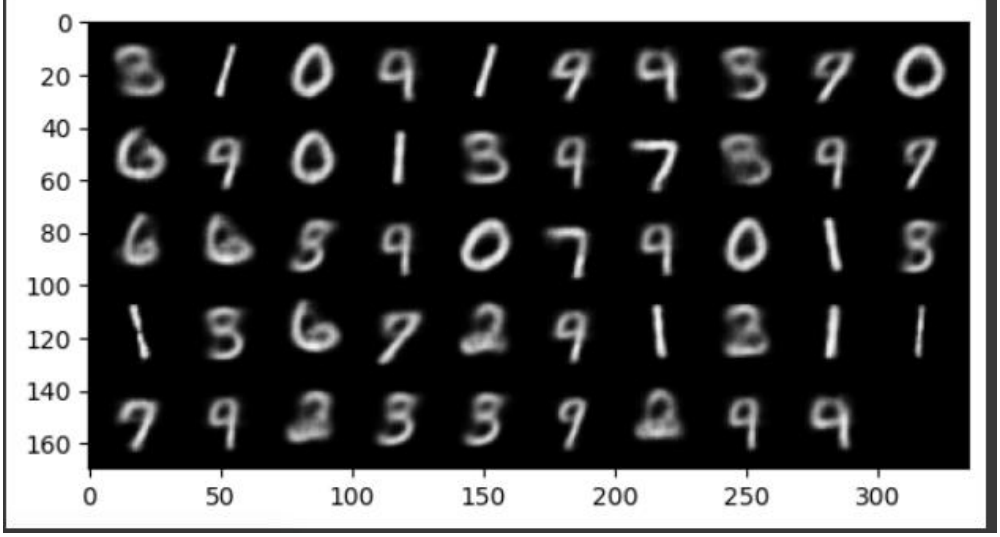



```
average reconstruction error: 19064.502707
```

Original images



VAE reconstruction:



در ادامه تابع interpolation رو داریم که این تابع وظیفه درون‌یابی (Interpolation) بین دو تصویر ورودی را بر عهده دارد. هدف این است که ویژگی‌های فضای نهفته (Latent Space) دو تصویر را ترکیب کنیم و یک تصویر جدید و ترکیبی بسازیم.

```
# TODO get both mu values for each image and use the
img1 = img1.to(device)
img2 = img2.to(device)
# interpolation of the two latent vectors
# TODO: write a weighted average based on lambda in
mu1, _ = model.encoder(img1)
mu2, _ = model.encoder(img2)
inter_latent = lambda1 * mu1 + (1 - lambda1) * mu2
```

ابتدا از آنجایی که ممکن است مدل روی GPU اجرا شود، تصاویر را به GPU انتقال می‌دهیم.

ورودی‌های img1 و img2 به مدل VAE داده می‌شوند.
تنها مقدار mu (میانگین توزیع فضای نهفته) را استخراج می‌کنیم و مقدار logvar را نادیده می‌گیریم.
در واقع به جای نمونه‌گیری تصادفی از فضای نهفته از مقدار میانگین فضای نهفته (mu) به عنوان نمونه استفاده می‌کنیم.

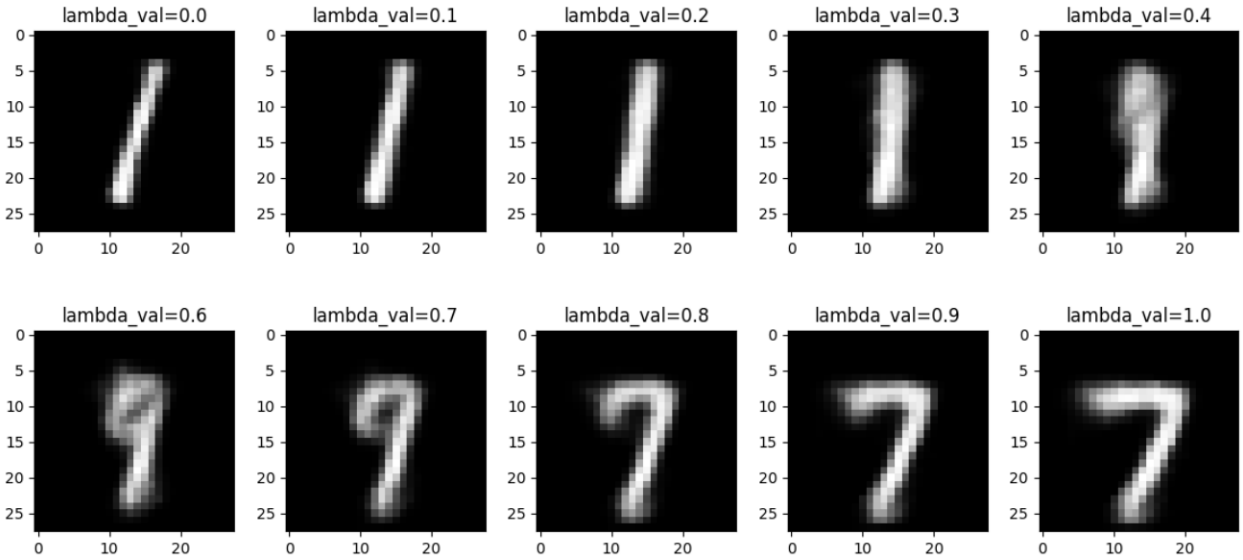
در نهایت درون‌یابی خطی بین mu1 و mu2 بر اساس وزن lambda1 انجام می‌شود:

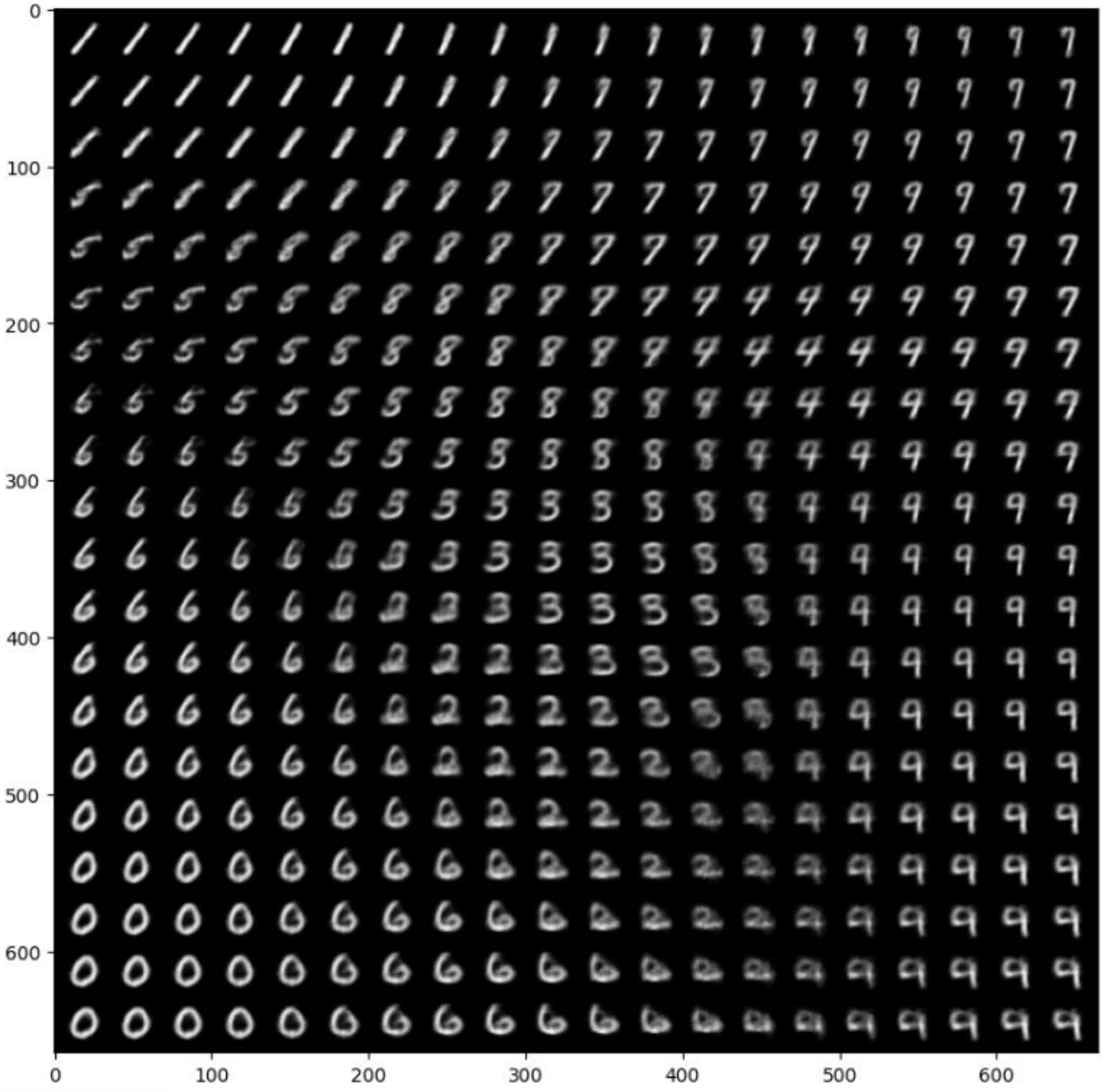
اگر lambda1 = 1 باشد: خروجی دقیقاً برابر mu1 خواهد شد.

اگر lambda1 = 0 باشد: خروجی دقیقاً برابر mu2 خواهد شد.

اگر lambda1 = 0.5 باشد: خروجی ترکیبی مساوی از mu1 و mu2 خواهد بود.

این تکنیک به ما اجازه می‌دهد تا ویژگی‌های دو تصویر را ترکیب کرده و یک تصویر جدید تولید کنیم.




0
100
200
300
400
500
600
0
100
200
300
400
500
600

بخش دوم

```
##############TODO################
negative_prompt = [""] * batch_size
negative_text_inputs = tokenizer(
    negative_prompt,
    padding="max_length",
    max_length=77,
    truncation=True,
    return_tensors="pt",
)
negative_text_input_ids = negative_text_inputs.input_ids

negative_prompt_embeds = text_encoder(negative_text_input_ids.cuda())
negative_prompt_embeds = negative_prompt_embeds[0]
negative_prompt_embeds = negative_prompt_embeds.to(dtype=prompt_embeds_dtype, device=device)
negative_prompt_embeds = negative_prompt_embeds.repeat(1, 1, 1)
negative_prompt_embeds = negative_prompt_embeds.view(batch_size * 1, max_length, -1)
#################################
```

این بخش از کد وظیفه تولید بردارهای embedding برای پرامپت‌های منفی را بر عهده دارد. در Stable Diffusion پرامپت منفی به مدل می‌گوید چه چیزی نباید در تصویر وجود داشته باشد.

ابتدا یک لیست negative_prompt با اندازه batch_size ساخته می‌شود. مقدار "" (رشته خالی) نشان‌دهنده یک پرامپت منفی خنثی است یعنی هیچ اطلاعات خاصی برای حذف وجود ندارد.

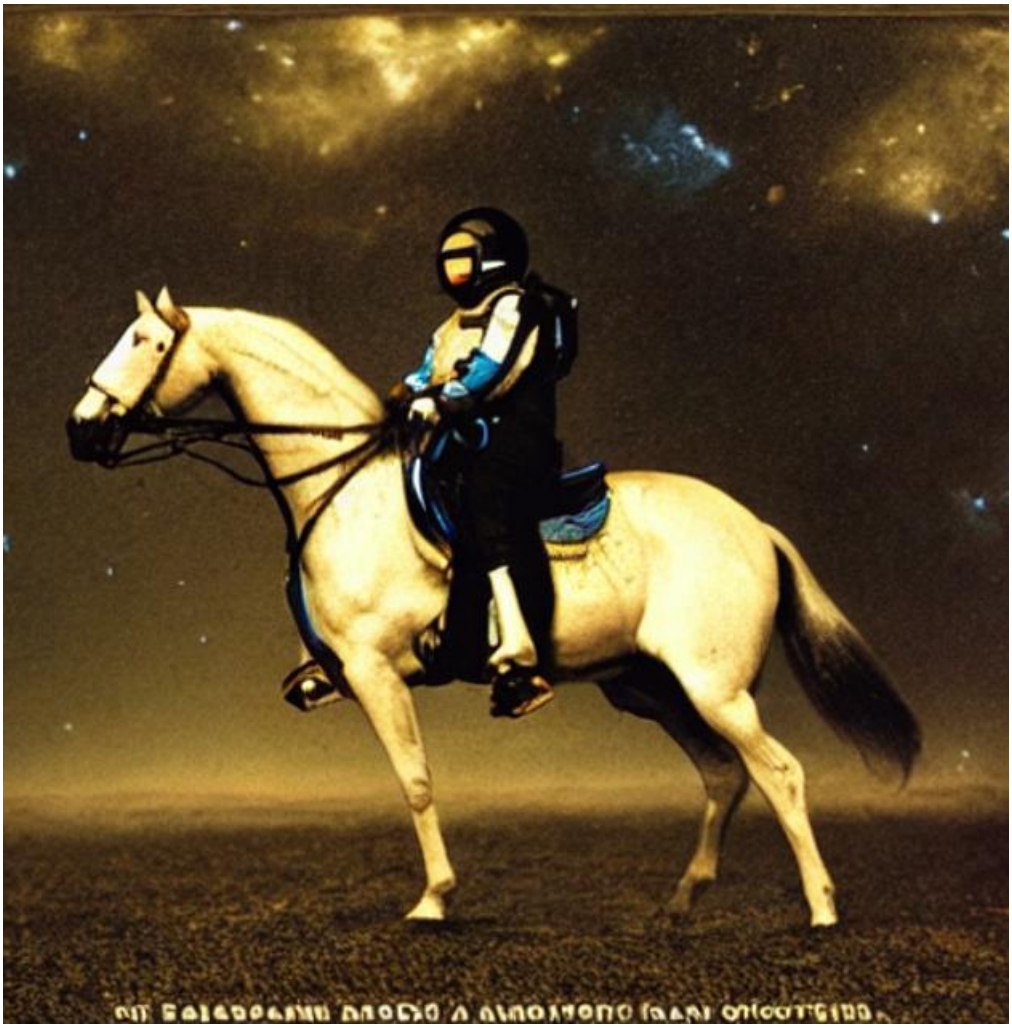
مانند پرامپت مثبت اینجا هم توکن‌های عددی تولید می‌شوند. از max_length=77 استفاده شده که مطابق با محدودیت مدل است. return_tensors="pt" باعث می‌شود خروجی یک تنسور PyTorch باشد.

در ادامه text_encoder توکن‌ها را به بردارهای ویژگی تبدیل می‌کند. خروجی text_encoder شامل چندین مقدار است، اما فقط قسمت اول یعنی negative_prompt_embeds[0] مورد نیاز است.

در ادامه نوع داده negative_prompt_embeds به همان دیتاتایپ (dtype) و به (device) منتقل می‌شود که قبلاً برای prompt_embeds استفاده شده بود.

در نهایت بردارها را در طول‌های خاصی تکرار می‌کند و ساختار داده را تغییر می‌دهد تا با prompt_embeds یکسان شود.

در نهایت به نتایج زیر رسیدیم:



محمد حقیقت – 403722042

برای رفع برخی ایرادات و ابهامات از Chatgpt استفاده شده است.