Deep Learning for Computer Vision

HW2

電子所 ICS 組, R13943015, 張根齊

Problem 1: Conditional Diffusion Models

1. Describe your implementation details and the difficulties you encountered.

A. Training setup:

1. BATCH_SIZE = 200
2. num_epochs = 50
3. Criterion: MSELoss
4. Optimizer: Adam
5. Learning rate scheduler:

```
scheduler = CosineAnnealingLR(optimizer, T_max=num_epochs * len(combined_loader), eta_min = 0) # update lr each batch
```

6. Data transformation(mean = std = [0.5, 0.5, 0.5]):

```
# Data Transformations
transform_train = transforms.Compose([
    transforms.Resize((28, 28)),
    transforms.ToTensor(),
    transforms.Normalize(mean=train_mean, std=train_std),
])
```

B. Model setting and beta scheduler:

1. Model setting:

The Unet model is adapted from https://github.com/byrkbrk/conditional-ddpm.

```
net = ContextUnet(in_channels=3, height=28, width=28, n_feat=128, n_cfeat=20, n_downs=2)
```

2. beta scheduler:

T = n_timestep = 350

```python
def beta_scheduler(n_timestep=1000, linear_start=1e-4, linear_end=2e-2):
    betas = torch.linspace(linear_start, linear_end, n_timestep, dtype=torch.float64)
    return betas
```

C. Training implementation:

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
6: **until** converged

Step 1. Randomly sample a batch of image x_0 from dataset $q(x_0)$. Since this is conditional generative model, we also need to get the label of image x_0. I combine two 10-classes datasets(MNISTM and SVHN) into one 20 classes dataset(n_cfeat = 20). Function *get_masked_context* randomly zero out one-hot label vector with probability = 0.1 which work as unconditional training.

```python
for images, label_dataset_image in tqdm(combined_loader, desc=f'Epoch {epoch + 1}/{num_epochs}', unit='batch'):  #進度條以batch完成度為趴數
    # ========================================================= step1: x0 ~ q(x0) ============================================================
    # x0 is original image: images
    dataset_label = label_dataset_image[0]
    image_label = label_dataset_image[1]
    images, dataset_label, image_label = images.to(device), dataset_label.to(device), image_label.to(device)# model, input output of model are moved to GPU during training
    N, C, H, W = images.shape[0], images.shape[1], images.shape[2], images.shape[3]
    # ========================================================= step1-1: preprocess of label ================================================
    # version6: combine two datasets into one
    image_label = image_label + 10 * dataset_label
    image_label = F.one_hot(image_label, num_classes=20)  # Encode class labels as one-hot vectors, shape = (N, 20)
    # Apply masking to the context (class labels)
    masked_image_label = get_masked_context(image_label).to(device) # shape = (N, 20) # randomly zero out one-hot vector: [0,0,1,0,..] -> [0,0,0,0,..]
```

Step 2. Sample t from Uniform distribution 1 ~ T, since python is 0-based indexing, sample t from 0 ~ T-1.

```python
    # ================================================= step2: t ~ Uniform({1,...,T}) ================================================
    t_np = np.random.randint(0, T, size=(N,))  # Sample t from Uniform({0, ..., T-1}) # python is 0-indexing
    t = torch.from_numpy(t_np)
    t = t.to(device)
```

Step 3. Sample noise from normal distribution.

```python
    # ================================================= step3: epsilon ~ N(0, 1) ================================================
    noise = torch.randn(N, C, H, W)  # Sample noise from normal distribution with mean 0 and standard deviation 1
    noise = noise.to(device)         # dtype = float32     # shape = N* C* H* W
```

Step 4. Take gradient descent step on noise ϵ and predicted noise ϵ_θ. Our model use noise $x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon$ and t to predict noise ϵ_θ.

x_t calculation:

```python
def noise_x_t(alpha_hat, x_0, t, epsilon):
    """given x[0](origin image) and t, return x[t](noisy image from x[0] by adding epsilon repeatly t times)"""
    """一步登天"""
    """epsilon.shape = x_0.shape = N*C*H*W,  t.shape = N"""
    """alpha_hat.shape = T, index t = 0 ~ T-1"""

    sqrt_alpha_hat = torch.sqrt(alpha_hat[t]).unsqueeze(1).unsqueeze(1).unsqueeze(1) # sqrt_alpha_hat.shape = N*1*1*1
    sqrt_one_minus_alpha_hat = torch.sqrt(1 - alpha_hat[t]).unsqueeze(1).unsqueeze(1).unsqueeze(1) # sqrt_one_minus_alpha_hat.shape = N*1*1*1

    x_t = sqrt_alpha_hat * x_0 + sqrt_one_minus_alpha_hat * epsilon

    return x_t #x[t] is N*C*H*W
```

predict noise and take gradient descent step:

```python
# ================================== step4: take gradient descent step on noise and predict_noise ==================================
x_t = noise_x_t(alpha_hat=alpha_hat, x_0=images, t=t, epsilon=noise)
x_t = x_t.to(device)

# 10% chance to zero-out # 10% of training is unconditional training
predict_noise = net(x=x_t.float(), t=(t/T), c=masked_image_label.float()) #x=N*C*H*W, t=N, y=N*20

loss = criterion(noise, predict_noise) # MSE loss
optimizer.zero_grad()
loss.backward()
optimizer.step()
```

D. Inference implementation:

Algorithm 2 Sampling

1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: **for** $t = T, \dots, 1$ **do**
3: $\quad \mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\quad \mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

Step 1. Sample x_T from normal distribution. Since this is conditional generative model, we need to embed label info into model.

```python
# ========================================================== step1: x_T ~ N(0, 1) ===========================================================
# x_t.shape = (N, C, H, W)
x_t = torch.randn( N , 3, image_size, image_size) # Sample x_T from normal distribution with mean 0 and std 1
x_t = x_t.to(device)                                    # dtype = float32     # shape = (N, C, H, W)

# --------------------- model input: y ---------------------
# labels_cond.shape = (N, 20)    # y is 0 ~ 19
y =  [torch.full((50,), each_digit) for each_digit in digit] # Create a tensor by repeating each element 50 times(concat each tensor with shape 50)
y = torch.cat(y, dim=0)   # Concatenate all tensors along dim=0, y.shape = N
labels_cond = F.one_hot(y, num_classes=num_classes).to(device)  # labels_cond.shape = (N, 20)
labels_uncond = torch.zeros_like(labels_cond)                   # labels_uncond.shape = (N, 20)
```

Step 2. Denoising from x_T to x_0. At each steps we minus predicted noise from x_t to get x_{t-1}. The method I embed label is adapted from Classifier-Free Diffusion Guidance (arXiv:2207.12598). I choose w_guide = 2.

$$\tilde{\boldsymbol{\epsilon}}_t = (1 + w)\boldsymbol{\epsilon}_\theta(\mathbf{z}_t, \mathbf{c}) - w\boldsymbol{\epsilon}_\theta(\mathbf{z}_t)$$

In step 2, sample z from a normal distribution to increase the model's variability. Then, predict the noise both conditionally and unconditionally. Combine them using the formula above, and update x_t using the predicted noise.

```python
# ================================================== step2: denoise for-loop ==================================================
for t_idx in range(T-1, -1, -1): # for t_idx = T-1,...,0  # since python is 0-indexing
    # ---------------------- generate z ----------------------
    if t_idx > 0:
        z = torch.randn((N, 3, image_size, image_size)) # Sample z from normal distribution with mean 0 and standard deviation 1
    else: # if add noise at last iteration, outcome even worse
        z = torch.zeros((N, 3, image_size, image_size))
    z = z.to(device)

    # ---------------------- model input: t ----------------------
    # t.shape = N    # t is 0 ~ T-1
    t = torch.full((N, ), t_idx).to(device)  # create a tensor with all element = t_idx and it's shape = N

    # ---------------------- Predict the noise ----------------------

    # Predict the noise epsilon(x_t, t) with "conditional" generate & "unconditional" generate
    if t_idx % 40 == 0:
        print(t_idx)
    noise_pred_cond = net(x_t.float(), (t/T).float(), labels_cond.float())      # shape = (N, C, H, W)
    noise_pred_uncond = net(x_t.float(), (t/T).float(), labels_uncond.float()) # shape = (N, C, H, W)

    # ref: Classifier-Free Diffusion Guidance(arXiv:2207.12598)
    # epsilon_x_t_t.shape = (N, C, H, W)
    epsilon_x_t_t = (1 + w) * noise_pred_cond - w * noise_pred_uncond # on device

    #sqrt_alpha[t_idx], sqrt_one_minus_alpha_hat[t_idx], one_minus_alpha[t_idx], sqrt_one_minus_alpha[t_idx] are scalars
    x_t = ((1/sqrt_alpha[t_idx]) * (x_t - (one_minus_alpha[t_idx]/sqrt_one_minus_alpha_hat[t_idx]) * epsilon_x_t_t) +
           sqrt_one_minus_alpha[t_idx] * z )# x_t is on device
```

E. Difficulty:

一開始我用的是另一個 open source 的 model(https://github.com/dome272/Diffusion-Models-pytorch?tab=readme-ov-file). 但這個 model 的 capacity 太低，導致訓練很多次正確率都上不去，之後改了另一個比較大的 Unet model，正確率就上去了。

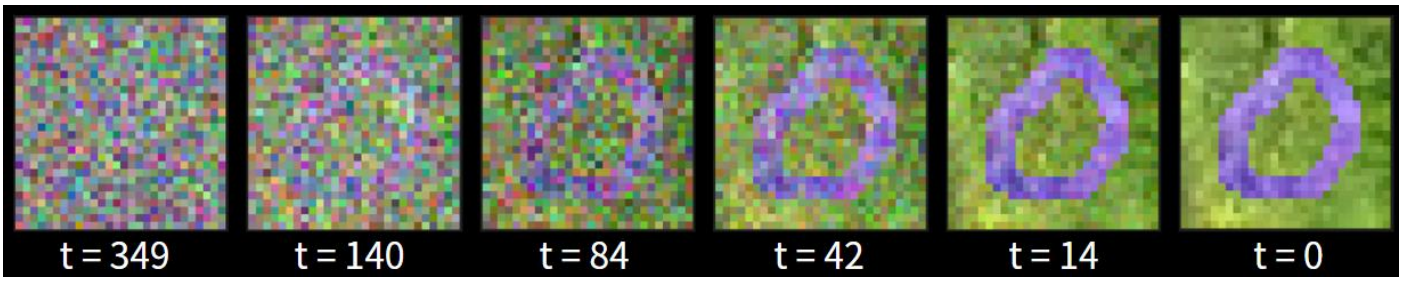
2. Please show 10 generated images for each digit (0-9) from both MNIST-M & SVHN dataset in your report. You can put all 100 outputs in one image with columns indicating different noise inputs and rows indicating different digits. [see the below MNIST-M example, you should visualize BOTH MNIST-M & SVHN].

A. MNIST-M

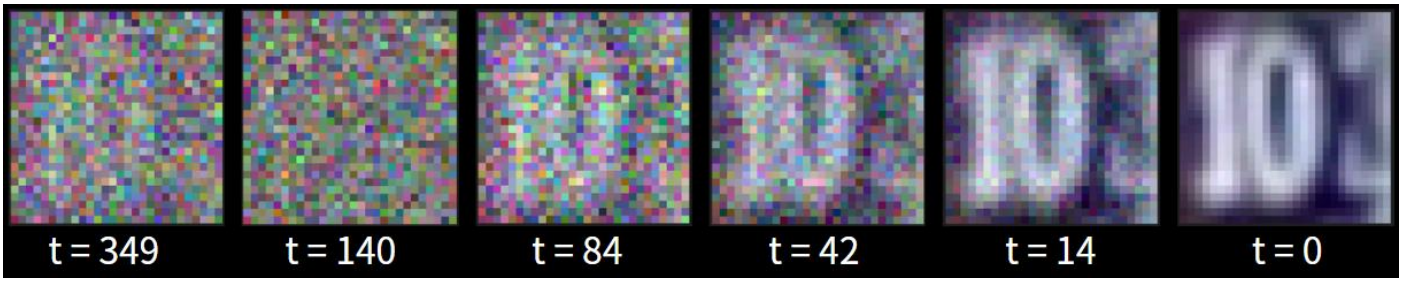
B. SVHN

3. Visualize a total of six images from both MNIST-M & SVHN datasets in the reverse process of the first “0” in your outputs in (2) and with different time steps. [see the MNIST-M example below, but you need to visualize BOTH MNIST-M & SVHN]

A. MNIST-M



B. SVHN



Problem 2: DDIM

1. Please generate face images of noise 00.pt ~ 03.pt with different eta in one grid. Report and explain your observation in this experiment.

$$\boldsymbol{x}_{t-1} = \sqrt{\alpha_{t-1}} \underbrace{\left(\frac{\boldsymbol{x}_t - \sqrt{1-\alpha_t}\epsilon_\theta^{(t)}(\boldsymbol{x}_t)}{\sqrt{\alpha_t}} \right)}_{\text{“ predicted } \boldsymbol{x}_0\text{”}} + \underbrace{\sqrt{1-\alpha_{t-1}-\sigma_t^2} \cdot \epsilon_\theta^{(t)}(\boldsymbol{x}_t)}_{\text{“direction pointing to } \boldsymbol{x}_t\text{”}} + \underbrace{\sigma_t \epsilon_t}_{\text{random noise}} \tag{12}$$

$$\sigma_{\tau_i}(\eta) = \eta \sqrt{(1-\alpha_{\tau_{i-1}})/(1-\alpha_{\tau_i})} \sqrt{1-\alpha_{\tau_i}/\alpha_{\tau_{i-1}}},$$

From the formula above (Denoising Diffusion Implicit Models: arXiv:2010.02502), η is a multiplier of σ. The larger η is, the greater the influence of noise on the update of x_t. When η=0, the random noise becomes 0, making the generation process deterministic. As shown in the figure above, when η increases, the generated image deviates more from the one generated with η=0.

2. Please generate the face images of the interpolation of noise **00.pt ~ 01.pt.** The interpolation formula is **spherical linear interpolation**, which is also known as **slerp**. in this case, α = {0.0, 0.1, 0.2, …, 1.0}.What will happen if we simply use linear interpolation? Explain and report your observation. (There should be **two images** in your report, one for spherical linear and the other for linear)

A. slerp

B. lerp

From the SLERP interpolation results, we can observe that the image generated from 00.pt gradually transforms into the image from 01.pt. However, the images generated using LERP clearly fail to achieve smooth transitions. I guess the reason SLERP works better may be it considers the latent space to be non-Euclidean, where meaningful features, such as those representing facial structures, are distributed non-linearly.

Problem 3: Personalization

1. Conduct the CLIP-based zero shot classification on the hw2_data/clip_zeroshot/val, explain how CLIP do this, report the accuracy and 5 successful/failed cases.

A. Accuracy: 56.48%

B. 5 successful cases:

File name	Predicted class	Ground truth class
34_493.png	castle	castle
40_481.png	couch	couch
35_454.png	boy	boy
24_458.png	house	house
27_488.png	bee	bee

C. 5 failed cases:

File name	Predicted class	Ground truth class
35_479.png	dolphin	boy
27_453.png	willow_tree	bee
3_460.png	willow_tree	rabbit
11_494.png	willow_tree	dinosaur
33_478.png	kangaroo	mouse

CLIP performs zero-shot classification by leveraging joint embeddings for images and text. It is pretrained on large-scale (image, text) pairs using contrastive learning. For zero-shot classification, the input image is encoded into feature latent by image encoder, and class labels are represented as text prompts (“A photo of a {object}.”), which are also encoded into embeddings by text encoder. The model computes similarities between the image feature latent and text embedding, and then selects the label with the highest similarity. This approach allows CLIP to recognize unseen class without fine-tuning.

2. What will happen if you simply generate an image containing multiple concepts (e.g., a <new1> next to a <new2>)? You can use your own objects or the provided cat images in the dataset. Share your findings and survey a related paper that works on multiple concepts personalization, and share their method.

A. prompt: “a <new1> in the style of <new2>"

I think some of the generated images are quite good, but some aren't that good. For example, the corgi's facial features in the bottom-left image are merged together, making them unclear. Additionally, some directly combined concept images lose logical consistency. For instance, in the bottom-right image, the corgi is standing on water.

B. Paper: Multi-Concept Customization of Text-to-Image Diffusion (arXiv:2212.04488)

In the paper, authors fine-tune multiple concepts simultaneously by using modifier tokens(represent personalized concept, ex: a specific dog or David Revoy). These tokens, initialized with rare words, are optimized within the cross-attention layers. By focusing on only key and value of the cross-attention layer, it enhances computational efficiency, memory efficiency and reducing overfitting risks.

For multi-concept integration, the model jointly trains on the datasets, which combines each dataset of target concepts. This selective fine-tuning strategy not only preserves the original model’s knowledge(w/o train from scratch) but also allows smooth composition of different elements, ensuring the personalized concepts are correctly represented without compromising performance.

Reference:

1. Chatgpt

https://chatgpt.com/

2. How To Train a Conditional Diffusion Model From Scratch

https://wandb.ai/capecape/train_sd/reports/How-To-Train-a-Conditional-Diffusion-Model-From-Scratch--VmlldzoyNzIzNTQ1

3. Diffusion Models | PyTorch Implementation

https://www.youtube.com/watch?v=TBCRlnwJtZU

4. Diffusion-Models-pytorch

https://github.com/dome272/Diffusion-Models-pytorch

5. conditional-ddpm

https://github.com/byrkbrk/conditional-ddpm

6. Classifier-Free Diffusion Guidance

arXiv:2207.12598

7. Hugging Face

https://github.com/huggingface/diffusers/tree/main/examples/textual_inversion

8. fastai

https://github.com/fastai/diffusion-nbs/blob/master/Stable%20Diffusion%20Deep%20Dive.ipynb

9. Multi-Concept Customization of Text-to-Image Diffusion

arXiv:2212.04488

10. Denoising Diffusion Implicit Models

arXiv:2010.02502