國立陽明交通大學 深度學習實驗 LAB4

313553036 多工所 1 葉軒宇

i. Derivate conditional VAE formula

$$p(x \mid c;\theta) = \int p(x \mid z, c;\theta)\, p(z \mid c)\, dz$$

θ : parameter
c : condition
z : latent code

⟶ rewrite as integral form and using chain rule of probability, it will became

$$\log p(x \mid c;\theta) = \log p(x, z \mid c;\theta) - \log p(z \mid x, c;\theta)$$

then adding arbitrary distribution $q(z \mid c)$ to both side and integral over z

$$\begin{aligned}
\log p(x \mid c;\theta) &= \int q(z \mid c) \log p(x \mid c;\theta)\, dz \\
&= \int q(z \mid c) \log p(x, z \mid c;\theta)\, dz \\
&\quad - \int q(z \mid c) \log p(z \mid x, c;\theta)\, dz \\
&= \left\{ \int q(z \mid c) \log p(x, z \mid c;\theta)\, dz \right. \\
&\quad \left. - \int q(z \mid c) \log q(z \mid c)\, dz \right\} \\
&\quad + \left\{ \int q(z \mid c) \log q(z \mid c)\, dz \right. \\
&\quad \left. - \int q(z \mid c) \log p(z \mid x, c;\theta)\, dz \right\} \\
&= L(x, q, \theta \mid c) + KL(q(z \mid c) \,\|\, p(z \mid x, c;\theta))
\end{aligned}$$

$$\because \text{①}\ L(x, q, \theta \mid c) = \int q(z \mid c) \log p(x, z \mid c; \theta)\, dz - \int q(z \mid c) \log q(z \mid c)\, dz$$

$$\text{②}\ KL(q(z \mid c) \,\|\, p(z \mid x, c; \theta)) = \int q(z \mid c) \log q(z \mid c)\, dz - \int q(z \mid c) \log p(z \mid x, c; \theta)\, dz$$

$$\text{By ①, ②} \Rightarrow L(x, q, \theta \mid c) = \log p(x \mid c; \theta) - KL(q(z \mid c) \,\|\, p(z \mid x, c; \theta))$$

③

cause ③ is true for all distribution of $q(z \mid c)$, so we can use another neural network as encoder

$$L(x, q, \theta \mid c) = \log p(x \mid c; \theta) - KL(q(z \mid x, c; \theta') \,\|\, p(z \mid x, c; \theta))$$

$$= \mathbb{E}_{z \sim q(z \mid x, c; \theta')}[\log p(x \mid z, c; \theta) + \log p(z \mid c) - \log q(z \mid x, c; \theta')]$$

$$= \mathbb{E}_{z \sim q(z \mid x, c; \theta')}[\log p(x \mid z, c; \theta)] - KL(q(z \mid x, c; \theta') \,\|\, p(z \mid c))$$

ii. Introduction

在 Lab4 中，我們使用 conditional VAE（CVAE）來預測影片幀。CVAE 模型被訓練來根據先前的幀和人體姿態預測下一個 frame。為了生成整個影片，模型會使用預測的 frame 作為下一次預測的輸入。為了訓練模型在當前 frame 的情形以預測下一個 frame，我們需要一個 latent variable 來表示當前 frame 中不存在的資訊。我們使用後驗預測器來預測在給定當前 frame 和人體姿態下的 latent variable 的後驗分佈。然後，我們使用預測的後驗分佈來抽樣 latent variable，該變數將用於預測下一個 frame。接者，我們訓練模型，使用前一步的 frame、當前 frame 的人體姿態和抽樣

的 latent variable 來預測下一個 frame。訓練目標是最小化預測的後驗分佈和真實後驗分佈之間的 KL divergence，以及預測幀和真實幀之間的 MSE loss。為了避免 KL disappear，我們在訓練過程中使用 KL annealing strategy 來增加 KL 權重。此外，為了防止模型使用來自先前 frame 的錯誤訊息，我們使用 teacher forcing strategy 來控制使用真實 frame 或預測 frame 作為輸入的比例。

iii. Implementation details

1. How do you write your training/testing protocol

   Training protocol:

```python
def training_one_step(self, img, label, adapt_TeacherForcing):
    img = img.permute(1, 0, 2, 3, 4)
    label = label.permute(1, 0, 2, 3, 4)
    out = img[0]

    reconstruction_loss = 0.0
    kl_loss = 0.0

    for i in range(1, self.train_vi_len):
        label_feat = self.label_transformation(label[i])
        if adapt_TeacherForcing:
            human_feat_hat = self.frame_transformation(img[i - 1])
        else:
            human_feat_hat = self.frame_transformation(out)

        z, mu, logvar = self.Gaussian_Predictor(human_feat_hat, label_feat)

        parm = self.Decoder_Fusion(human_feat_hat, label_feat, z)
        out = self.Generator(parm)

        reconstruction_loss += self.mse_criterion(out, img[i])
        kl_loss += kl_criterion(mu, logvar, batch_size=self.batch_size)

    beta = torch.tensor(min(self.kl_annealing.get_beta(), 1.0), dtype=torch.float32).to(self.args.device)
    loss = reconstruction_loss + beta * kl_loss

    self.optim.zero_grad()
    loss.backward()
    self.optimizer_step()

    return loss
```

   先將第一幀、label 輸入，再將它們的輸出傳入 Gaussian Predictor。再將得到的 mu、log var、z 與原先的 frame 及 label 輸入 decoder，之後由 generator 得到 output。根據使用 teacher forcing 與否來決定用何者做為下一輪的輸入。在全部的 frame 都輸入後再將 beta 與 reconstruction loss 相加，最終與 kl_loss 相乘以更新 loss。

   Testing protocol:

```python
def val_one_step(self, img, label, idx=0):
    img = img.permute(1, 0, 2, 3, 4)  # change tensor into (seq, B, C, H, W)
    label = label.permute(1, 0, 2, 3, 4)  # change tensor into (seq, B, C, H, W)
    print(f"Input img shape: {img.shape}")
    print(f"Input label shape: {label.shape}")
    assert label.shape[0] == 630, "Testing pose segence should be 630"
    assert img.shape[0] == 1, "Testing video segence should be 1"

    decoded_frame_list = [img[0].cpu()]
    label_list = []

    # Initialize the first frame
    out = img[0]

    for i in range(1, 630):
        f = self.frame_transformation(decoded_frame_list[i - 1].to(self.args.device))
        l = self.label_transformation(label[i]).to(self.args.device)
        z, _, _ = self.Gaussian_Predictor(f, l)
        z = torch.randn_like(z)
        decoded = self.Decoder_Fusion(f, l, z)
        decoded = self.Generator(decoded)
        decoded_frame_list.append(decoded.cpu())
        label_list.append(l.cpu())

    # Please do not modify this part, it is used for visulization
    generated_frame = stack(decoded_frame_list).permute(1, 0, 2, 3, 4)
    label_frame = stack(label_list).permute(1, 0, 2, 3, 4)

    print(f"Generated frame shape: {generated_frame.shape}")
    assert generated_frame.shape == (1, 630, 3, 32,64), \
        f"The shape of output should be (1, 630, 3, 32, 64), but your output shape is {generated_frame.shape}"

    self.make_gif(generated_frame[0], os.path.join(self.args.save_root, f'pred_seq{idx}.gif'))

    # Reshape the generated frame to (630, 3 * 64 * 32)
    generated_frame = generated_frame.reshape(630, -1)

    print(f"Shape of generated_frame: {generated_frame.shape}")
    return generated_frame
```

一開始先對前一幀還有當前的 label 進行 transformation，再使用 Gaussian predictor 來生成 z，並用隨機 noise 替換它。之後將特徵、標籤、z 融合，透過 generator 來生成新的 frame。最後將生成的 frame 加入 list。

2. How do you implement reparameterization tricks?

```python
def reparameterize(self, mu, logvar):
    std = torch.exp(0.5 * logvar)
    epsilon = torch.randn_like(std)
    z = mu + std * epsilon
    return z
```

由 logvar 得到標準正態分佈(std)，再從 std 中抽樣一個隨機變量，將這兩個數值相乘後加上 mu。因為將隨機性與 network 分離，這樣使得整個模型變得可以微分。

3. How do you set your teacher forcing strategy?

```
# Teacher Forcing strategy
parser.add_argument( *name_or_flags: '--tfr', type=float, default=0, help="The initial teacher forcing ratio")
parser.add_argument( *name_or_flags: '--tfr_sde', type=int, default=10, help="The epoch that teacher forcing ratio start to decay")
parser.add_argument( *name_or_flags: '--tfr_d_step', type=float, default=0, help="Decay step that teacher forcing ratio adopted")
parser.add_argument( *name_or_flags: '--ckpt_path', type=str, default=None, help="The path of your checkpoints")
```

```
def teacher_forcing_ratio_update(self):
    if self.current_epoch >= self.tfr_sde and self.tfr > 0:
        self.tfr -= self.tfr_d_step
    self.tfr = max(self.tfr, 0)
```

先根據參數決定是否使用 teacher forcing，若要使用則於訓練時每次減少 tfr_d_step。

4. How do you set your kl annealing ratio?

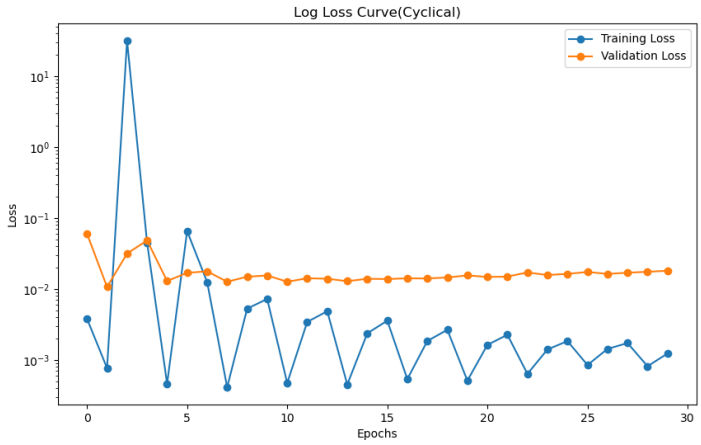
```
2 usages
class kl_annealing:
    def __init__(self, args, current_epoch=0):
        self.kl_anneal_type = args.kl_anneal_type
        self.kl_anneal_cycle = args.kl_anneal_cycle
        self.kl_anneal_ratio = args.kl_anneal_ratio
        self.beta = 0.001
        if self.kl_anneal_type == "Without":
            self.beta = 1.0

    1 usage
    def update(self, current_epoch):
        if self.kl_anneal_type == "Cyclical":
            self.beta = self.frange_cycle_linear(
                current_epoch,
                start=0.0,
                stop=1.0,
                n_cycle=self.kl_anneal_cycle,
                ratio=self.kl_anneal_ratio,
            )
        elif self.kl_anneal_type == "Monotonic":
            self.beta = self.frange_monotonic(
                current_epoch,
                start=0.0,
                stop=1.0,
                n_cycle=self.kl_anneal_cycle,
                ratio=self.kl_anneal_ratio,
            )
    3 usages
    def get_beta(self):
        return self.beta

    1 usage
    def frange_cycle_linear(self, n_iter, start=0.0, stop=1.0, n_cycle=1, ratio=1):
        new_beta = (n_iter % n_cycle) / n_cycle * ratio
        new_beta = round(new_beta, 5)
        new_beta = min(new_beta, stop)
        return new_beta

    1 usage
    def frange_monotonic(self, n_iter, start=0.0, stop=1.0, n_cycle=1, ratio=1):
        delta = ratio / n_cycle
        new_beta = round(n_iter * delta, 5)
        new_beta = min(new_beta, stop)
        return new_beta
```

根據參數使用的不同，會調用 frange_cycle_linear 或 frange_monotonic 兩個不同的函數。frange_cycle_linear 根據 step 來逐步增加 beta 值，anneal_cycle 決定其週期大小，進行循環改變 beta 值的效果。frange_monotonic 則是逐步遞增大小，在抵達設定值後即保持，不進行循環。

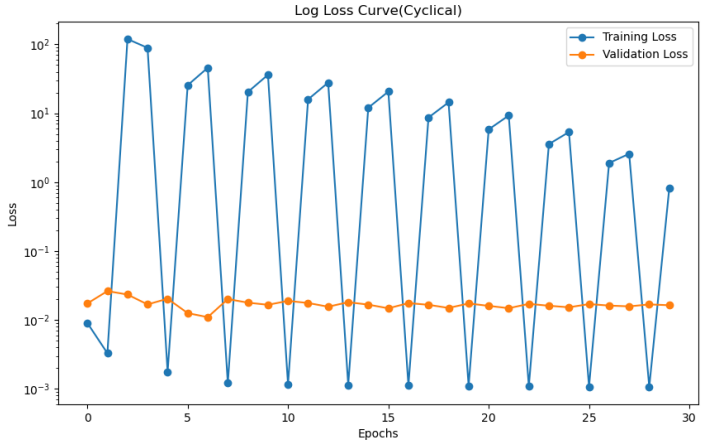
iv. Analysis & discussion

1. Plot Teacher forcing ratio
   a. Analysis & compare with the loss curve

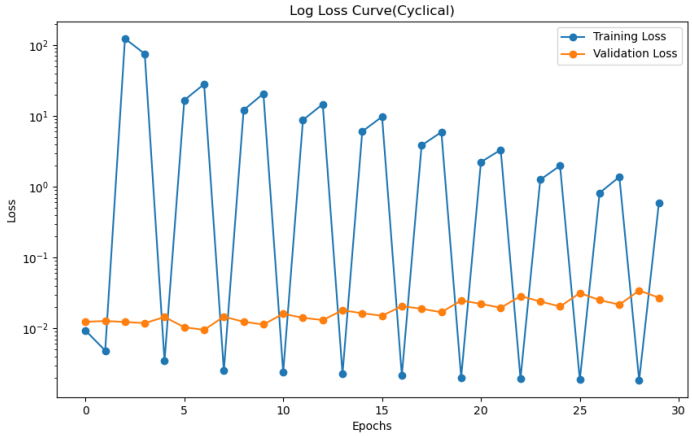
Tfr = 0.5, Tfr_sde = 5, Tfr_d_step = 0.1:



Tfr = 0.1, Tfr_sde = 30, Tfr_d_step = 0:

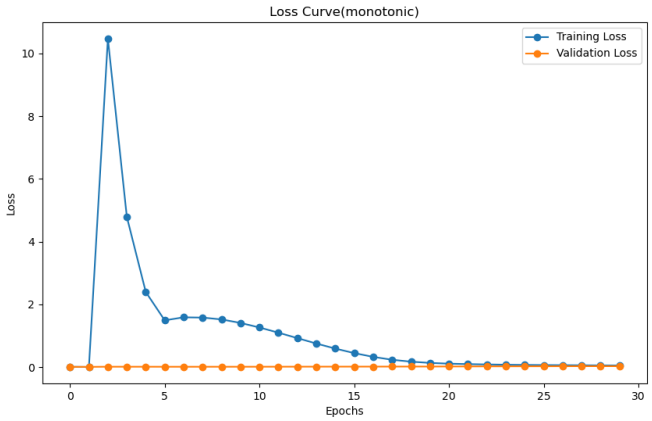


Tfr = 0, Tfr_sde = 30, Tfr_d_step = 0:



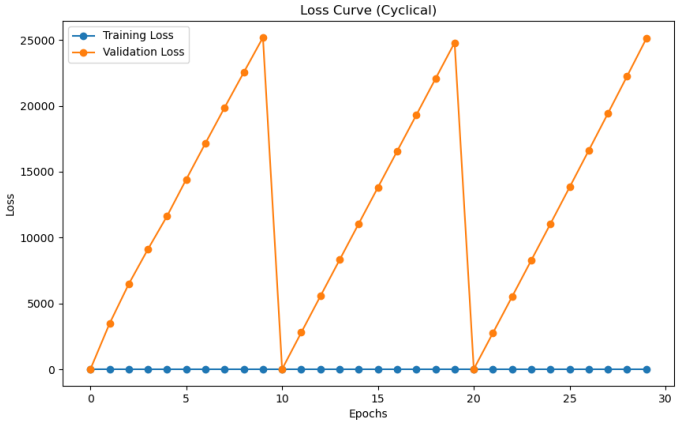
從這三張圖表可以看出，較高的 teacher forcing 比例（如 tfr = 0.5）有助於在訓練初期穩定模型的損失，且訓練損失的波動較小，但驗證損失仍然較高。而當 teacher forcing 比例較低或完全關閉時，訓練損失的波動增大，但驗證損失更加穩定，表明模型的泛化能力可能有所提高。

2. Plot the loss curve while training with dirrerent settings.
   a. With KL annealing(Monotonic)



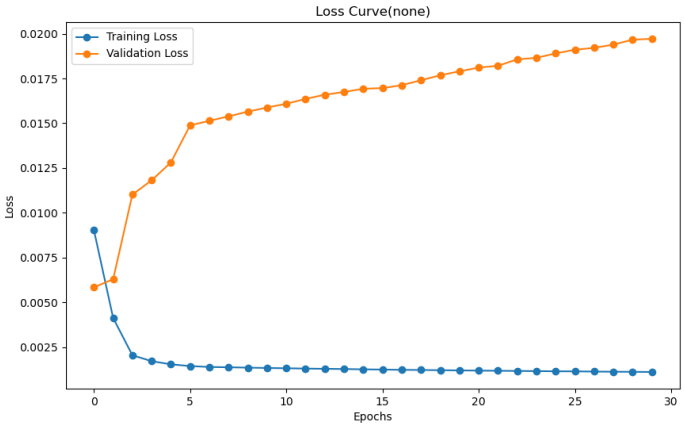
monotonic KL annealing 會使 KL loss 隨著訓練進行單調增長，這意味著模型更好地遵循正則化條件。在這種設定下，模型在初期迅速收斂，訓練和驗證損失最終都趨於穩定，顯示出較好的收斂效果。這樣的方法可能有助於模型更快地學習，但也可能導致過早收斂。

   b. With KL annealing(Cyclical)



cyclical KL annealing 使得 KL loss 隨著周期變化，這會在一定程度上強迫模型在某些 epoch 更加注重 KL loss 的正則化效果，並在其他時候減少其影響。這導致了驗證損失的劇烈波動。這樣的設定可能適合某些需要周期性學習的情形，但從圖表中看出，這種策略對模型的穩定性影響較大，並且可能不太適合這種應用場景。

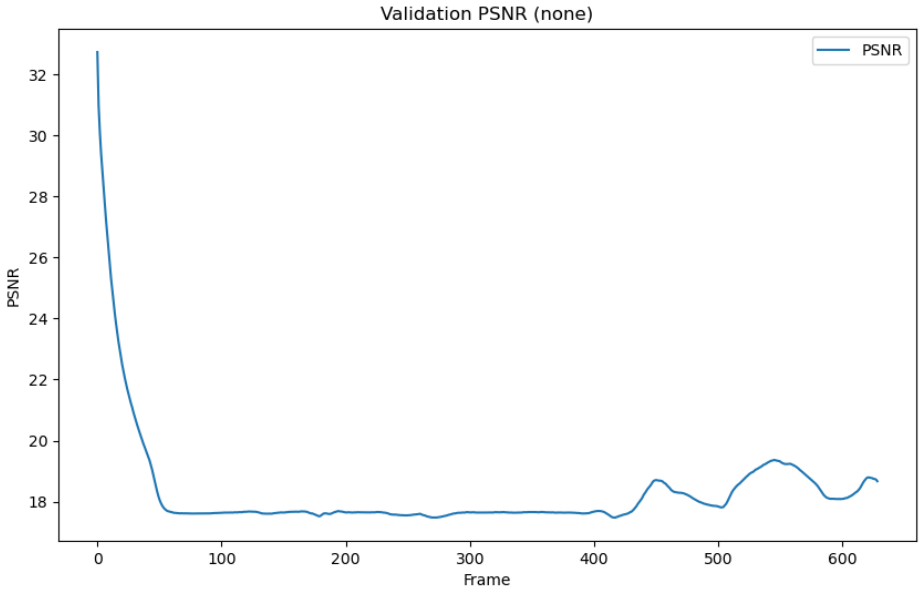
c. Without KL annealing

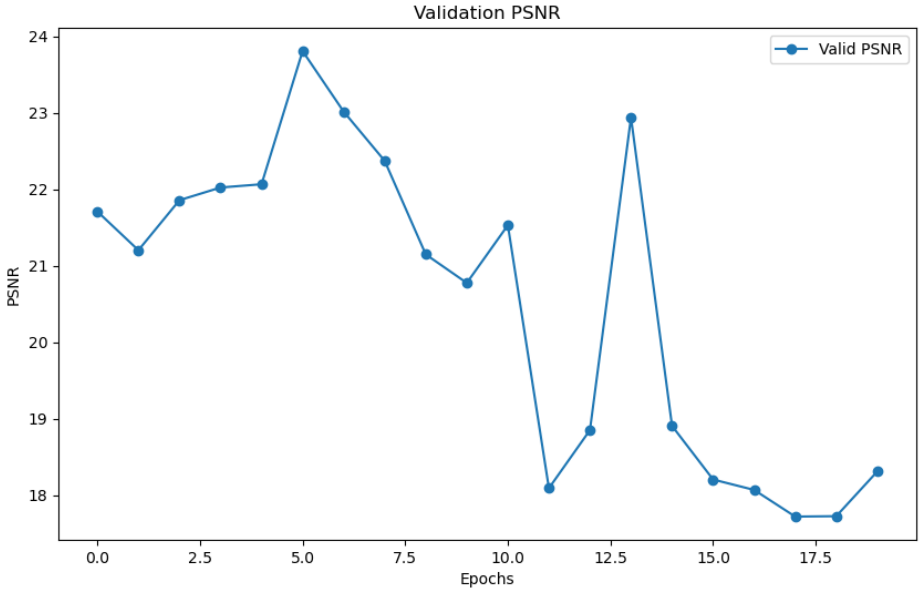


當不使用 KL annealing 時，KL loss 的權重保持不變（可能是 1 或更低），這可能導致模型過度擬合訓練數據。結果顯示，雖然訓練損失迅速降低並保持低值，但驗證損失不斷增加，這表明模型的泛化能力受到影響，出現了過擬合現象。

3. Plot the PSNR-per frame diagram in validation dataset

kl_annealing_type = none, kl_annealing_cycle = 30, kl_annelaing_ratio = 1
Tfr = 0, Tfr_sde = 5, Tfr_d_step = 0





Fast_train = 20, max_norm = 0.75

因為並沒有使用teacher forcing strategy，所以在模型一開始的下降幅度比較大。在400frame開始可能因為隨著影片生成的進展，模型的誤差開始累積，導致生成frame的波動。