Lab4: Conditional VAE for Video Prediction

系級:智能系統　學號:312581006　姓名:張宸瑋

1. Introduction

這次 lab 要實作基於 VAE 的模型的條件式視頻預測，在這次的實作主要用到兩個在 Paper[1][2]提出的概念，第一個是透過 GAN 模型，並且使用遇訓練的姿勢估計網路生成的姿勢圖像，以及前一幀，來使模型輸出具有一定質量的下一幀，而第二個是透過基於 VAE 的模型，並且結合 LSTM 以及 RNN 的方式預測未來幀，通過使用兩個參考幀，使模型有能力預測接下來的未來幀。

以下是本次實驗模型的主要架構，左邊的圖為訓練模型的過程，大致如下，Posterior Predictor透過當前幀輸入到frame encoder以及骨架圖像輸入到pose encoder的特徵圖，作為輸入的生成分布。而Generator以當前標籤，跟生成的上一幀畫面，以及由Posterior Predictor預測的分佈所抽樣的噪音作為輸入，生成當前的畫面。而在test的部分噪音會直接從先驗分佈作抽樣。

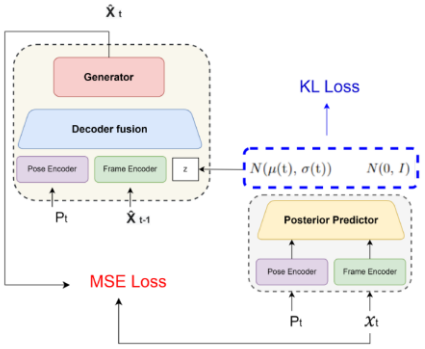


Figure 2. (a)

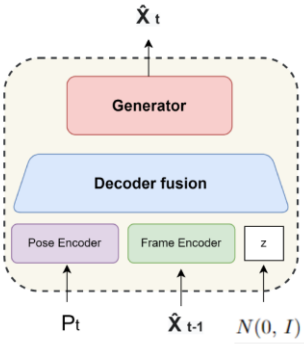


Figure 2. (b)

模型架構

而這次我們也需要在 KL 損失函數中添加一個可變的權重 beta，來調整 KL loss 影響的比例，透過不同的策略 Cyclical、Monotonic、Without KL annealing strategy 來調整 beta，並且訓練模型以及分析結果。

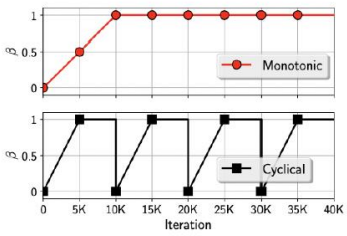


Figure 5. weight in different KL annealing strategies

週期性策略以及單調策略示意圖

2. Implementation details

A. How do you write your training protocol

```python
def training_stage(self):
    for i in range(self.args.num_epoch):
        train_loader = self.train_dataloader()
        total_loss = 0
        total_kld = 0
        loss = 0

        for (img, label) in (pbar := tqdm(train_loader, ncols=120)):
            adapt_TeacherForcing = True if random.random() < self.tfr else False
            img = img.to(self.args.device)
            label = label.to(self.args.device)
            loss, mse, kld = self.training_one_step(img, label, adapt_TeacherForcing)
            total_loss += mse.item() * img.size(0)
            total_kld += kld.item() * img.size(0)
            #total_kld
            beta = self.kl_annealing.get_beta()

            if adapt_TeacherForcing:
                self.tqdm_bar('train [TeacherForcing: ON, {:.2f}], beta: {:.2f}'.format(self.tfr, beta), pbar, loss, lr='{:.0e}'.format(self.scheduler.get_last_lr()[0]))
            else:
                self.tqdm_bar('train [TeacherForcing: OFF, {:.2f}], beta: {:.2f}'.format(self.tfr, beta), pbar, loss, lr='{:.0e}'.format(self.scheduler.get_last_lr()[0]))

        epoch_loss = total_loss/len(train_loader.dataset)
        epoch_kld = total_kld/len(train_loader.dataset)
        self.train_loss.append(epoch_loss)
        self.train_kld.append(epoch_kld)
        self.eval()
        self.scheduler.step()
        self.teacher_forcing_ratio_update()
        self.kl_annealing.update()
        print(f"Epoch {self.current_epoch} AVG MSE loss is {epoch_loss} and AVG KLD loss is {epoch_kld}")
        if self.current_epoch % self.args.per_save == 0:
            self.save(os.path.join(self.args.save_root, f"epoch={self.current_epoch}.ckpt"))
        self.current_epoch += 1
```

Training Stage

```python
def training_one_step(self, img, label, adapt_TeacherForcing):
    # TODO
    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)
    decoded_frame_list = [img[0].cpu()]
    mse = 0
    kld = 0
    self.label_transformation.zero_grad()
    self.frame_transformation.zero_grad()
    self.Gaussian_Predictor.zero_grad()
    self.Decoder_Fusion.zero_grad()
    self.Generator.zero_grad()
    human_feat_hat_list = []
    for i in range(0,self.train_vi_len):
        human_feat_hat = self.frame_transformation(img[i])
        human_feat_hat_list.append(human_feat_hat)
    for i in range(1, self.train_vi_len):
        #如果採用teacherforcing,encoder input為i-1張原圖,反之,使用預測出的圖
        label_feat = self.label_transformation(label[i])
        z, mu, logvar = self.Gaussian_Predictor(human_feat_hat_list[i],label_feat)
        if adapt_TeacherForcing:
            human_feat_hat = human_feat_hat_list[i-1]
        else:
            human_feat_hat = self.frame_transformation(decoded_frame_list[i-1].to(self.args.device))

        parm = self.Decoder_Fusion(human_feat_hat, label_feat, z)
        out = self.Generator(parm)
        mse += self.mse_criterion(out, img[i])
        kld += kl_criterion(mu, logvar, self.batch_size)
        decoded_frame_list.append(out.cpu())

    beta = self.kl_annealing.get_beta()
    loss = mse + kld * beta
    loss.backward()
    self.optimizer_step()
    return loss.detach().cpu().numpy() / self.train_vi_len, mse.detach().cpu().numpy() / self.train_vi_len, kld.detach().cpu().numpy() / self.train_vi_len
```

Training One Step

解說:

這段主要是訓練模型的程式碼，在訓練時每個 batch 的資料會被當作參數，輸入到 train_one_step 函數裡面，並且透過 adapt_TeacherForcing 判斷 Frame Encoding 的輸入是要用 Generator 生成的還是原本的前一個畫面，並且根據 self.train_vi_len 來決定要預測的影片序列的長度，並且依序將每個畫面輸出。

B. How do you implement reparameterization tricks

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

reparameterization tricks

解說:

這段程式碼主要是透過重新參數化的技巧，來實現反向傳播的計算。

C. How do you set your teacher forcing strategy

```python
def teacher_forcing_ratio_update(self):
    if self.current_epoch < self.args.tfr_sde:
        return 1.0

    steps_since_tfr_sde = self.current_epoch - self.args.tfr_sde
    if steps_since_tfr_sde >= 0 and steps_since_tfr_sde % 5 == 0:
        self.tfr -= self.args.tfr_d_step
    return self.tfr
```

teacher forcing strategy

解說:

這段程式碼主要是在描述我的 teacher forcing strategy，一開始會先判斷現在的 epoch 是否小於設定的 self.args.tfr_sde，如果還沒 ratio 維持 1，當 epoch 大於 self.args.tfr_sde 後，每五個 epoch 透過 self.args.tfr_d_step 依序更新 ratio。

D. How do you set your kl annealing ratio

```python
class kl_annealing():
    def __init__(self, args, current_epoch=0):
        # TODO
        self.args = args
        self.kl_anneal_type = args.kl_anneal_type
        self.current_epoch = current_epoch
        self.n_cycle = args.kl_anneal_cycle
        self.ratio = args.kl_anneal_ratio
        self.n_epoch = args.num_epoch
        self.L =  np.ones(self.n_epoch) * 1.0
        if self.kl_anneal_type == "Cyclical":
            self.frange_cycle_linear(n_epoch=self.n_epoch, n_cycle=self.n_cycle,ratio=self.ratio)
        elif self.kl_anneal_type == "Monotonic":
            self.frange_cycle_linear(n_epoch=self.n_epoch, n_cycle=1,ratio=self.ratio)

    def update(self):
        # TODO
        self.current_epoch+=1

    def get_beta(self):
        # TODO
        return self.L[self.current_epoch]

    def frange_cycle_linear(self, start=0.0, stop=1.0, n_epoch=0, n_cycle=4, ratio=0.4):
        # TODO
        period = n_epoch/n_cycle
        step = (stop-start)/(period*ratio) # linear schedule
        for c in range(n_cycle):
            v , i = start , 0
            while v <= stop and (int(i+c*period) < n_epoch):
                self.L[int(i+c*period)] = v
                v += step
                i += 1
        return self.L
```

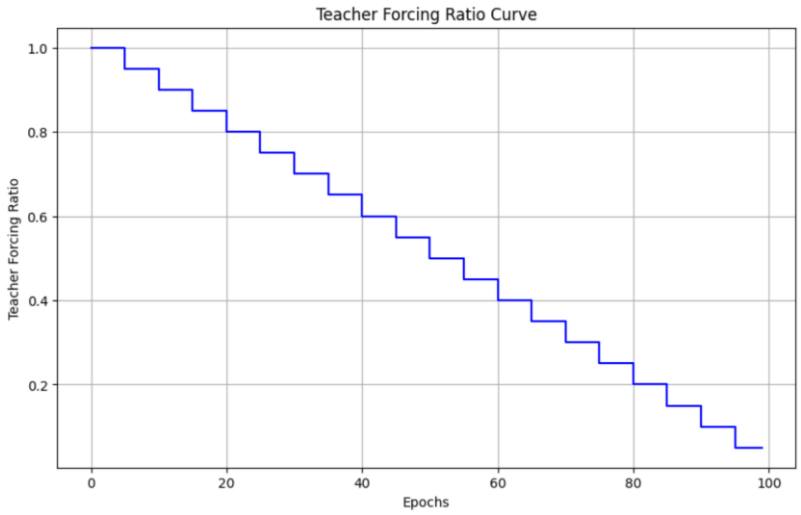
kl annealing ratio

解說:

這段程式碼主要是實現 KL annealing，透過逐漸增加權重，來幫助訓練模型，程式碼會根據 self.kl_anneal_type 來決定權重的更新是單一性還是週期性。

3. Analysis & Discussion

A. Plot Teacher forcing ratio

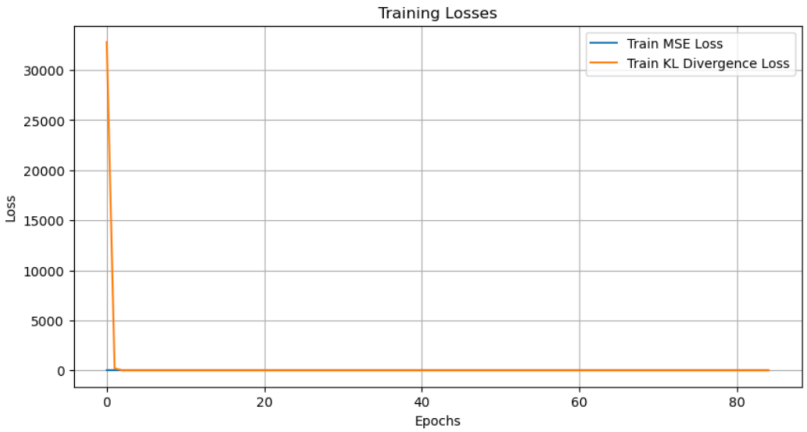


Teacher forcing ratio

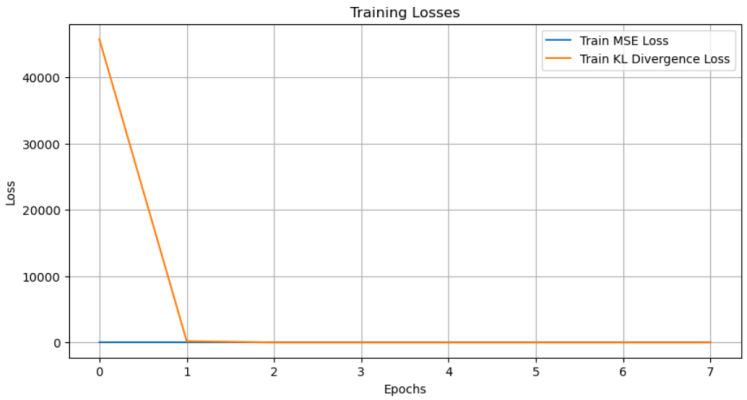
討論:

我認為透過老師在上課所講的策略，明顯的提升了我們訓練的結果，因為在訓練初期，如果我們直接使用預測的畫面來當作輸入，可能會導致後面模型沒辦法生成正確的序列，導致生成序列的不穩定，因此透過調整 ratio，並讓它線性遞減，可以讓訓練前期，透過原本的圖像來當作輸入，使模型學習到正確的重建損失。

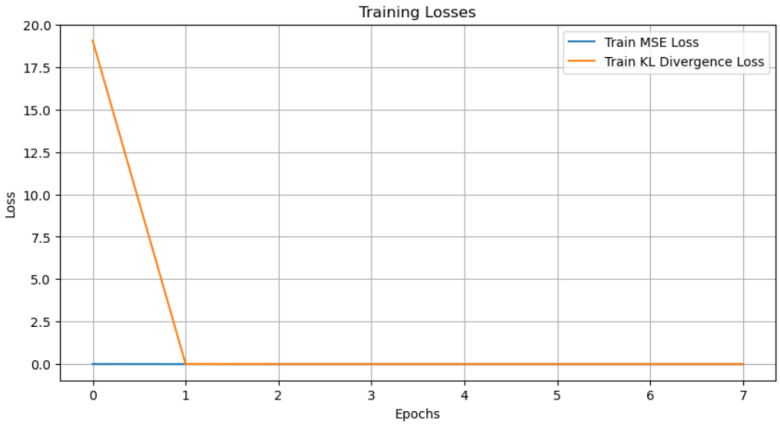
B. Plot the loss curve while training with different settings



With KL annealing (Cyclical)



With KL annealing (Monotonic)

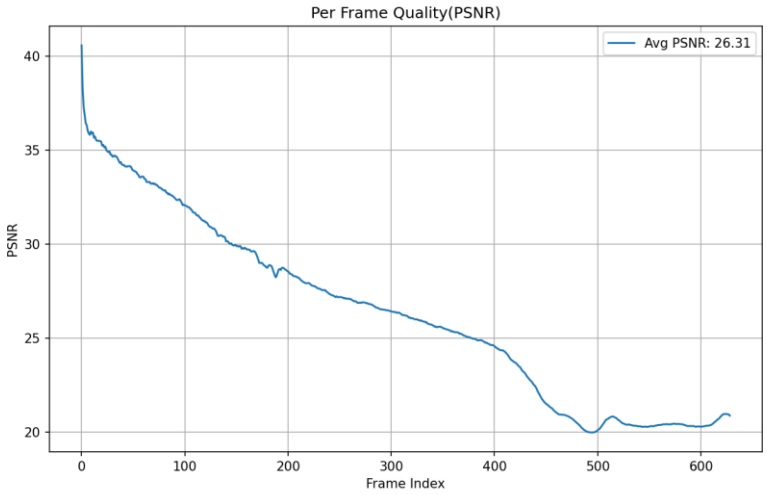


Without KL annealing

討論:

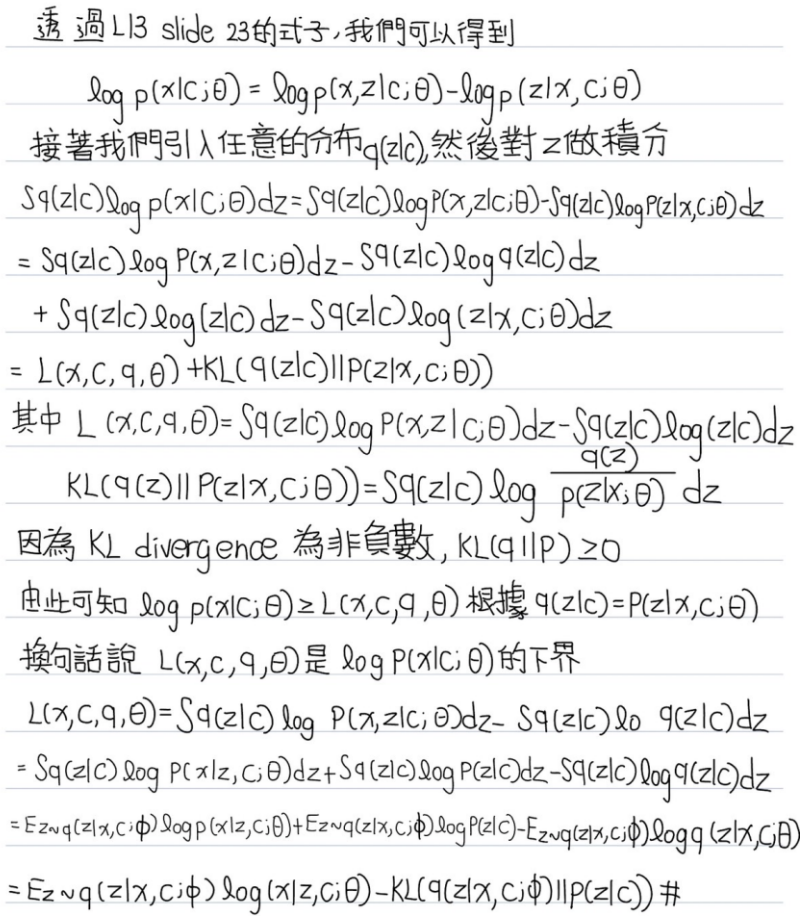
在這裡我們可以發現，當沒有採用 KL 退火策略的時候，代表模型相較於前兩種策略來說，會更關注 KL 散度的損失，因此我們可以看到圖中沒有採用 KL 退火策略的 KL loss 一開始是相對較小的，但這可能會導致模型生成的樣本的品質受到限制，而週期性 KL 退火策略，想對於單一性來說，因為它交替調整 KL 散度的權重，因此可以在訓練的時候平衡重建損失以及 KL 散度損失，而單一性策略，因為在後期更關注 KL 散度的損失，可能導致訓練的早期就限制了模型的生成能力。

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



PSNR-per frame diagram in validation dataset

D. Derive conditional VAE formula



conditional VAE formula

4. Discussion

這次的實驗我認為算是目前遇到最困難的，除了要讀熟兩篇 paper 以外，在訓練的部分也有許多細節需要注意，而當中讓我較有印象的部分是，當時把 train_loader 的 shuffle 設為 True，是希望提升模型的泛化能力，但效果好像不是如我預期的那樣，好幾次前幾個 epoch 在計算 loss 時，很容易就變成 nan，這裡我猜想可能是因為 dataset 的資料是有時間先後關係的，因此打亂造成了訓練的不穩定性。而在訓練的過程中，也讓我了解到要訓練像 VAE 這種網路是相對複雜的，因為它要優化的，不是單一個損失函數，因此在超參數的調整方面，是需要非常多的時間去做調整，才能達到理想的結果。