NYCU DL Lab4 Conditional VAE for Video Prediction

313551073 顏琦恩

I. Derivate conditional VAE formula

To learn a conditional distribution $p(X|c)$, we hope to maximize the marginal distribution $p(X|z,c;\theta)$, which means to maximize the log-likelihood

$$\log p(X|z,c;\theta) = \log p(X,z,c;\theta) - \log p(z|X,c;\theta)$$

we can rewrite the equation to the following term by applying an arbitrary distribution $q(z|c)$ on both sides

$$\int q(z|c) \log p(X|z,c;\theta)\, dz$$

$$= \int q(z|c) \log p(X,z,c;\theta)\, dz - \int q(z|c) \log p(z|X,c;\theta)\, dz$$

$$= \int q(z|c) \log p(X,z,c;\theta)\, dz - \int q(z|c) \log q(z|c)\, dz$$

$$+ \int q(z|c) \log q(z|c)\, dz - \int q(z|c) \log p(z|X,c;\theta)\, dz$$

$$= \mathcal{L}(X,c,q,\theta) + KL\big(q(z|c) \,\|\, p(z|X,c;\theta)\big)$$

$$\mathcal{L}(X,c,q,\theta) = \int q(z|c) \log p(X|z,c;\theta)\, dz - KL\big(q(z|c) \,\|\, p(z|X,c;\theta)\big)$$

then we can rewrite the equation into the following term:

$$\mathcal{L}(X,c,q,\theta)$$

$$= E_{z \sim q(z|X,c;\phi)} \log p(X|z,c;\theta) - KL\big(q(z|X,c;\phi) \,\|\, p(z|c)\big)$$

II. Introduction

In this lab, I use a Variational Autoencoder (VAE) to predict short videos. By inputting several pose images and a reference image, we can generate a short video with the man in the reference image doing the same poses as the input pose images. With the implementation of the training and testing process and several training methods such as kl annealing and teacher forcing strategy, we can improve the quality of produced video.

III. Implementation details

A. How do you write your training/testing protocol

In my training section, I applied the teacher forcing strategy to decide whether to use the ground truth to be the input of the frame encoder or not. Also,

After training, I use validation datasets to check the PSNR of each epoch to save the better checkpoint. In the validation part, I also applied the same method because the PSNR method can only compute 1 sequence at the same time.

In the testing section, I predict sequence batch by batch for a shorter processing time.

```python
    # python Trainer.py --DR ../LAB4_Dataset --save_root ./saved_models
    def training_one_step(self, img, label, adapt_TeacherForcing):
        # TODO
        # (B, seq, C, H, W)
        batch_size = img.shape[0]
        beta = self.kl_annealing.get_beta()

        kl_loss = 0.0
        mse_loss = 0.0
        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)

        decoded_frame = [img[0]]
        for i in range(1, self.args.train_vi_len):
            if adapt_TeacherForcing:
                prev_x = img[i-1]
            else:
                prev_x = decoded_frame[i-1]

            encoded_prev_x = self.frame_transformation(prev_x)
            encoded_p = self.label_transformation(label[i])

            encoded_x = self.frame_transformation(img[i])
            z, mu, logvar = self.Gaussian_Predictor(encoded_x, encoded_p)
```

```python

            x_hat = self.Generator(self.Decoder_Fusion(encoded_prev_x, encoded_p, z))
            decoded_frame.append(x_hat)

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

        loss = mse_loss + beta * kl_loss # seq loss

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

        return loss / batch_size    # avg seq loss
```

Fig. 1 Details of training code

```python
    # python Trainer.py --DR ../LAB4_Dataset --save_root ./saved_models --ckpt_path ./saved_models/epoch=0.ckpt --test
    def val_one_step(self, img, label):
        # TODO
        batch_size = img.shape[0]
        batch_psnr = []
        beta = self.kl_annealing.get_beta()

        kl_loss = 0.0
        mse_loss = 0.0
        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)

        decoded_frame = [img[0]]
        for i in range(1, self. args.val_vi_len):
            encoded_prev_x = self.frame_transformation(decoded_frame[i-1])
            encoded_p = self.label_transformation(label[i])

            encoded_x = self.frame_transformation(img[i])
            z, mu, logvar = self.Gaussian_Predictor(encoded_x, encoded_p)

            x_hat = self.Generator(self.Decoder_Fusion(encoded_prev_x, encoded_p, z))
            decoded_frame.append(x_hat)

            kl_loss += kl_criterion(mu, logvar, batch_size)
            mse_loss += self.mse_criterion(x_hat, img[i])
            batch_psnr.append(Generate_PSNR(x_hat, img[i]).item())

        loss = mse_loss + beta * kl_loss # seq loss

        return loss / batch_size, batch_psnr
```

Fig. 2 Details of validation code

```python
        # TODO
        for i in range(1, self.args.val_vi_len):
            encoded_x = self.frame_transformation(decoded_frame_list[i-1].to(self.args.device))
            encoded_p = self.label_transformation(label[i]).to(self.args.device)

            z, mu, logvar = self.Gaussian_Predictor(encoded_x, encoded_p) # get shape of z
            z = torch.randn_like(z) # sample from normal distribution

            x_hat = self.Generator(self.Decoder_Fusion(encoded_x, encoded_p, z))
            decoded_frame_list.append(x_hat.cpu())
            label_list.append(encoded_p.cpu())
```

Fig. 3 Details of testing code

B. How do you implement reparameterization tricks

From the paper Auto-Encoding Variational Bayes, we can see an example of implementing the reparameterization trick on page 5, so I just applied the formula in the paper to implement the reparameterization trick, but since the input of sigma is a log variance, I use the exponential to get its standard deviation. The following figures show the formula in the paper and my code of reparameterization.

$$\mathbf{z}^{(i,l)} = \boldsymbol{\mu}^{(i)} + \boldsymbol{\sigma}^{(i)} \odot \boldsymbol{\epsilon}^{(l)} \quad \text{and} \quad \boldsymbol{\epsilon}^{(l)} \sim \mathcal{N}(0, \mathbf{I})$$

Fig. 4 Formula of reparameterization trick in the paper

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

Fig. 5 Detail of reparameterization trick

C. How do you set your teacher forcing strategy

I applied the teacher forcing strategy in my training section. It is on if the random number is smaller than the initial teacher forcing ratio (tfr), and off otherwise. I used a linear function to determine the teacher forcing ratio, which is: tfr = tfr - d_step. The d_step represents the decay step of tfr, which is set to be 0.1 here. The tfr starts decaying from the 10th epoch beginning at a value of 1.0. When the teacher forcing strategy is on, I used the given image as the previous frame to predict the next frame, on the contrary, I used the predicted frame as the previous frame.

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

Fig. 6 Detail of the updating of the teacher forcing strategy

```python
def training_one_step(self, img, label, adapt_TeacherForcing):
    # TODO
    # (B, seq, C, H, W)
    batch_size = img.shape[0]
    beta = self.kl_annealing.get_beta()

    kl_loss = 0.0
    mse_loss = 0.0
    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)

    decoded_frame = [img[0]]
    for i in range(1, self.args.train_vi_len):
        if adapt_TeacherForcing:
            prev_x = img[i-1]
        else:
            prev_x = decoded_frame[i-1]

        encoded_prev_x = self.frame_transformation(prev_x)
        encoded_p = self.label_transformation(label[i])
```

Fig. 7 Implementation of the teacher forcing strategy in the training section

D. How do you set your kl annealing ratio

For the kl annealing ratio, I referenced from the paper Cyclical Annealing Schedule: A Simple Approach to Mitigating KL Vanishing.

```python
class kl_annealing():
    def __init__(self, args, current_epoch=0):
        # TODO
        self.current_epoch = current_epoch

        if args.kl_anneal_type == 'Cyclical':
            self.beta = self.frange_cycle_linear(args.num_epoch, 0.0, 1.0, args.kl_anneal_cycle, args.kl_anneal_ratio)
        elif args.kl_anneal_type == 'Monotonic':
            self.beta = self.frange_cycle_linear(args.num_epoch, 0.0, 1.0, 1, args.kl_anneal_ratio)
        else:
            self.beta = np.ones(args.num_epoch)

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

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

    def frange_cycle_linear(self, n_iter, start=0.0, stop=1.0,  n_cycle=1, ratio=1):
        # TODO refer to https://github.com/haofuml/cyclical_annealing/tree/master
        beta = np.ones(n_iter)
        period = n_iter/n_cycle # 每經過period個iteration，beta從0開始算
        step = (stop-start)/(period*ratio) # beta每次增加的量

        for c in range(n_cycle):
            v, i = start, 0
            while v <= stop and (int(i+c*period) < n_iter):
                beta[int(i+c*period)] = v
                v += step
                i += 1
        return beta
```

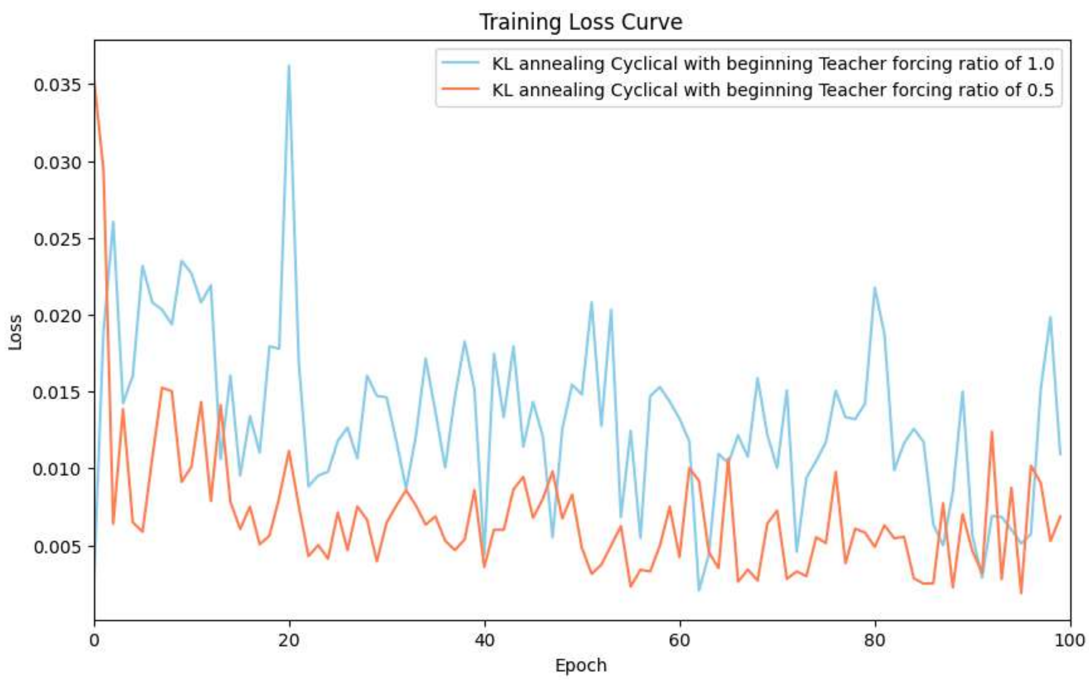
Fig. 8 Implementation of the kl annealing

IV. Analysis & Discussion

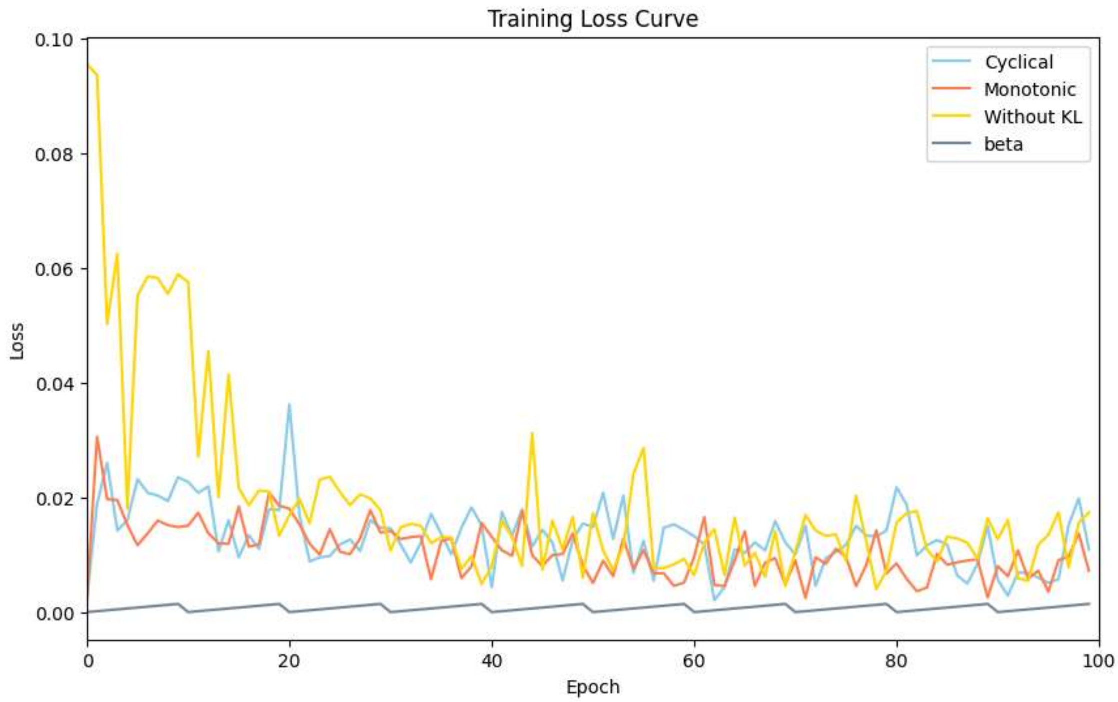
A. Plot Teacher forcing ratio

a. Analysis & compare with the loss curve

This experiment is using Cyclical annealing with the teacher forcing strategy beginning with the teacher forcing ratio of 1.0 and 0.5, to the end of 0.0. We can observe that using a lower beginning ratio of teacher forcing leads to a more stable loss curve in this lab.

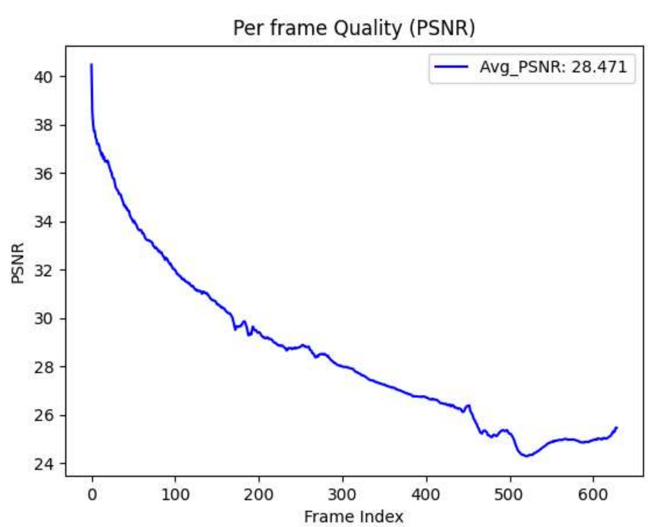


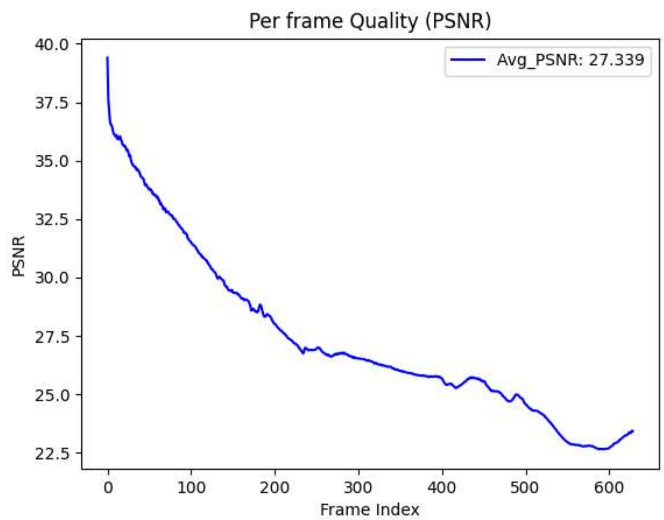
B. Plot the loss curve while training with different settings. Analyze the difference between them



This experiment is using the teacher forcing strategy beginning with the teacher forcing ratio of 1.0 to the end of 0.0. We can see that different kl annealing strategies have a small impact on the loss curve.

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

Cyclical	

Monotonic	
Without kl annealing