Deep Learning Lab4: Conditional VAE for Video Prediction

ID: 412551030 Name: 黃羿寧

1. Introduction

In this lab, we were requested to train a model that predicts and generates the future video frame based on the pose information using a modified VAE-based video generation model[1]. Furthermore, teacher forcing strategy, reparameterization tricks and derivations of conditional VAE were asked to be listed in detailed for deeper understanding of the VAE. Additionally, two kinds of KL annealing methods (Monotonic, Cyclical) mentioned in Fu et al. (2019)[2] were applied to the model training, aiming to avoid KL vanish issues during the training process. As the result, the peak signal to noise ratio (PSNR) of validation dataset and testing dataset shows that the model with highest predictive performance of 26 & 23 was trained using monotonic KL annealing methods.

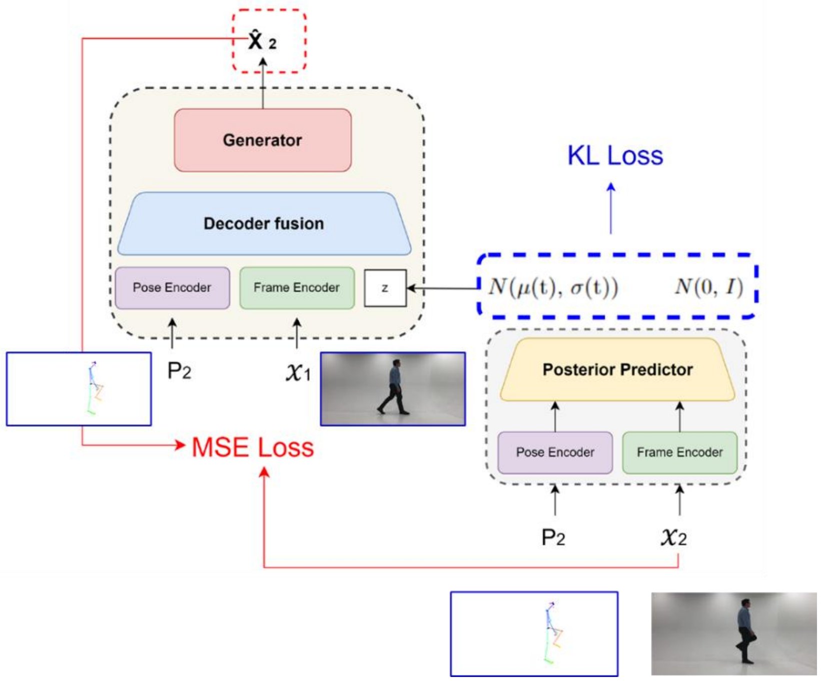

Figure 1. The overall architecture of CVAE model in this lab

2. Implementation details

a. How do you write your training protocol (10%)

In this lab, we followed the previous study[1] to train the CVAE model. In the previous study, the stochastic video generation model with a fixed prior over the latent variables (SVG-FP) predicted the next frame using encoder, a prior distribution, LSTM and a decoder. However, we replace the LSTM to NN-based Posterior predictor, Decoder fusion and Generator in this part. In the training phase of the model, each batch of data is transformed to iterate in the order of frames. The first frame (0th frame) is used as the input to the Decoder in the first iteration. The next frame (1st frame) along with a pose image serves as the input for the Encoder. This process generates a distribution from which the KL divergence is computed. Utilizing the reparameterization trick, a latent variable z is sampled. Next, the latent variable (z), the predicted output, and the pose image are input into the Decoder to generate the predicted frame. This predicted frame is then compared with the ground truth to compute the reconstruction error which is mean squared error (MSE). Note that teacher forcing strategy was applied to this lab, the input of the Decoder to generate the next frame was adjusted to the ground truth in some cases (TF = 1). Finally, the sum of the KL divergence and the MSE gives the Evidence Lower Bound (ELBO), which is used to optimize the model. This procedure is designed to balance the reconstruction accuracy (measured by MSE) and the probability model's fidelity (measured by KL Divergence), aiming to produce frames that are not only accurate but also plausible according to the learned distribution. In the `training_one_step` function, the Encoder and Gaussian Predictor calculates the KLD and the Decoder Fusion and the Generator calculates the MSE. And the update of the model depends on the ELBO variable.

```
    def training_one_step(self, img, label, adapt_TeacherForcing):
        # TODO
        self.train()
        img = img.permute(1, 0, 2, 3, 4)
        label = label.permute(1, 0, 2, 3, 4)

        avg_ELBO, avg_KLD, avg_MSE = 0.0, 0.0, 0.0
        pre_out = img[0]
        self.optim.zero_grad()
        for i in range(1, self.train_vi_len):
            # Encoder
            human_feat = self.frame_transformation(img[i])
            label_feat = self.label_transformation(label[i])
            z, mu, logvar = self.Gaussian_Predictor(human_feat,
label_feat)
            KLD = kl_criterion(mu, logvar, self.batch_size)

            # Decoder
            out_feat = self.frame_transformation(pre_out)
            parm = self.Decoder_Fusion(out_feat, label_feat, z)
            out = self.Generator(parm)
            MSE = self.mse_criterion(out, img[i])

            #Update predictive frame
            pre_out = img[i] if adapt_TeacherForcing else out
            ELBO = MSE + (KLD * self.kl_annealing.get_beta())

            avg_ELBO += ELBO
            avg_KLD += KLD
            avg_MSE += MSE

        avg_ELBO = avg_ELBO/(self.train_vi_len - 1)
        avg_MSE = avg_MSE/(self.train_vi_len - 1)
        avg_KLD = avg_KLD/(self.train_vi_len - 1)

        avg_ELBO.backward()
        self.optimizer_step()

        return avg_ELBO, avg_MSE, avg_KLD
```

b. How do you implement reparameterization tricks?

The reparameterization trick was proposed aiming to move out the random process out of model's training pathway. This adjustment is done by incorporating a Gaussian node to replace the original random node (z), which is not differentiable and limited the backpropagation. This Gaussian noise is then scaled and shifted by the predicted mean and standard deviation to produce the sample. By doing do, the sampling step becomes differentiable, enabling the computation of gradients and thereby facilitating effective training.

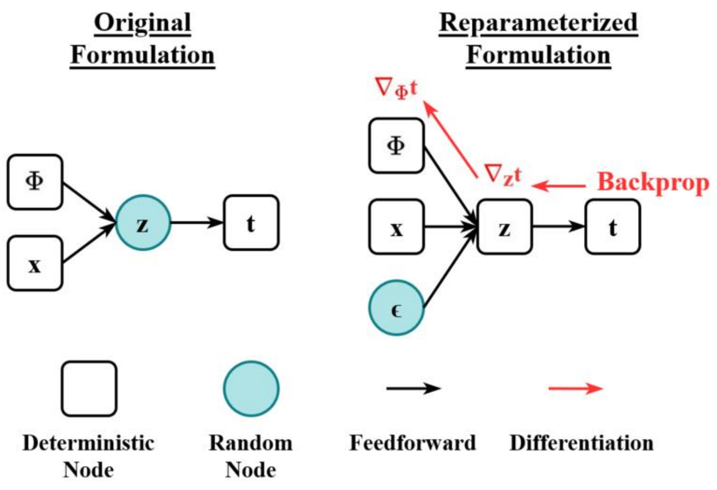

Figure 2. The illustration of reparameterization tricks

The function of reparameterization is shown here. The standard deviation from the logarithm of the variance requires two-step calculation: exponential and square root. Therefore, $std = sqrt(exp(logvar)) = exp(0.5 * logvar)$

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

c. How do you set your teacher forcing strategy (5%)

Teacher forcing is a technique used to help models more accurately predict the next frame, especially during the early stages of training. When a model starts training, it might not quickly understand which information from the current input is necessary to predict the next frame. This can lead to errors in model training. If the model continuously uses the predicted previous frame as the input for predicting the current frame, these errors can accumulate, reducing the efficiency and effectiveness of the model's learning. To address this, teacher forcing replaces the decoder's input with the actual ground truth during training. This helps the model better understand the information required between two frames, thus improving learning accuracy and speed.

In this process, the teacher forcing ratio was set between 0-1. The default tfr_sde is 10, and one tfr_d_step is 0.1, so the tfr will be decayed to 0 in the 20 epochs in this case.

```
    def teacher_forcing_ratio_update(self):
        # TODO
        if self.current_epoch >= self.tfr_sde:
            self.tfr = max(self.tfr - self.tfr_d_step, 0.0)
```

However, whether the mechanism switch on depends on the dice. If the random value in this batch is not above the pre-defined tfr, the teacher forcing will not be initiated.

```
adapt_TeacherForcing = True if random.random() < self.tfr else False
```

Input of the Encoder in the next prediction will be adjusted to the ground truth if TeacherForcing is on.

```
#Update predictive frame
pre_out = img[i] if adapt_TeacherForcing else out
```

d. How do you set your kl annealing ratio (5%)

KL annealing is used to solve the KL vanishing issues often encountered when training VAEs. The KL divergence is used to maintain the distribution differences between the latent and the prior gaussian, so when the model over relies on the reconstruction error, this model might not achieve the goal of VAE but a simple AE. This often happens in the beginning of the training. To address this, KL annealing introduces a parameter, β, which ranges from 0 to 1. This method gradually (monotonic) or periodically (cyclical) increases the contribution of the KL loss, allowing the model to focus first on reducing the reconstruction loss. As β increases back to 1, the full loss is considered, enhancing the impact of the KL loss. This technique helps balance the learning process, ensuring that both the reconstruction and the distributional aspects of the model are adequately learned.

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

        if self.args.kl_anneal_type == 'None':
            self.schedule = np.ones(args.num_epoch)
        elif self.args.kl_anneal_type == 'Monotonic':
            self.schedule = self.frange_cycle_linear(args.num_epoch,
n_cycle=1, ratio=args.kl_anneal_ratio)
        elif self.args.kl_anneal_type == 'Cyclical':
            self.schedule = self.frange_cycle_linear(args.num_epoch,
n_cycle=args.kl_anneal_cycle, ratio=args.kl_anneal_ratio)

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

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

    def frange_cycle_linear(self, n_iter, start=0.0,
stop=1.0,  n_cycle=1, ratio=1):
        # TODO
        L = np.ones(n_iter)
        period = n_iter / n_cycle
        step = (stop-start)/((period-1) * ratio)

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

3. Analysis & Discussion

a. Plot Teacher forcing ratio

The teacher forcing and loss of the model using monotonic KL annealing strategy was shown in Figure 3. The loss (ELBO) significantly decreases with the teacher forcing strategy.

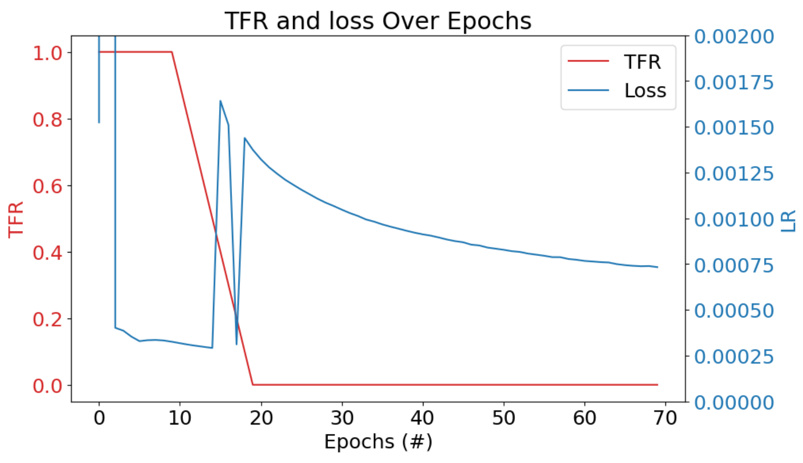

Figure 3. The TFR and loss in the model training

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

The KL annealing methods were not significant influencing the loss curve. Instead, the teacher forcing mechanism decreases the loss in three strategies. However, it was still able to see the cyclical strategy in the fluctuated loss curve (Figure 4), and this result was similar to the beta employed in the training (Figure 5).

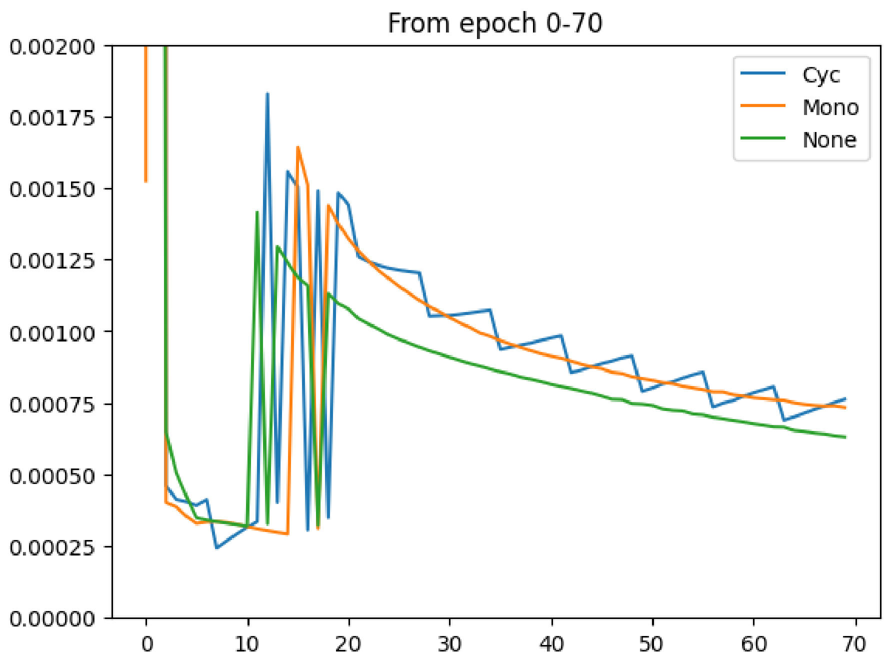

Figure 4. The loss curve between different KL annealing strategy

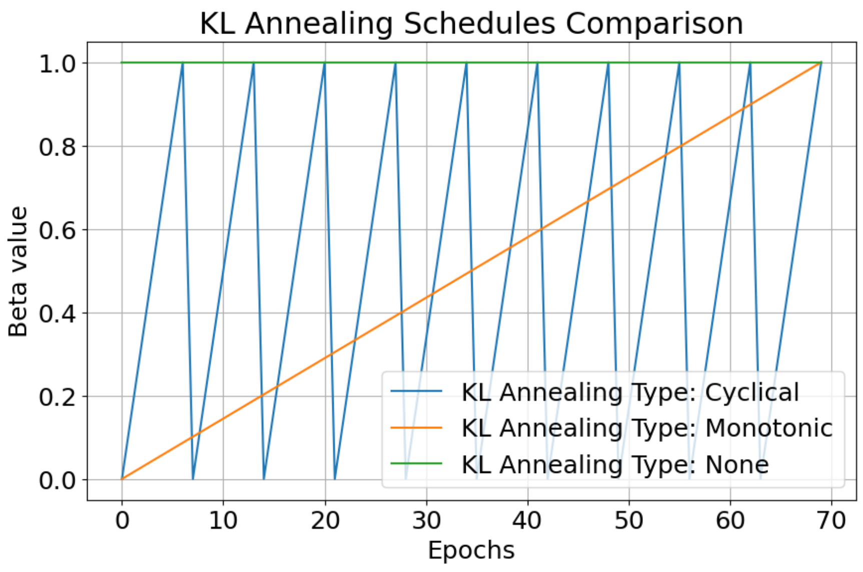

Figure 5 The beta between different KL annealing strategy

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

I used the last epoch of each model to test on the validation dataset. As a result shown in Figure 6, it's obvious to see that the monotonic model has the highest predictive effectiveness even in the 400+ frame. However, the other model declines significantly in the first 200 frame.

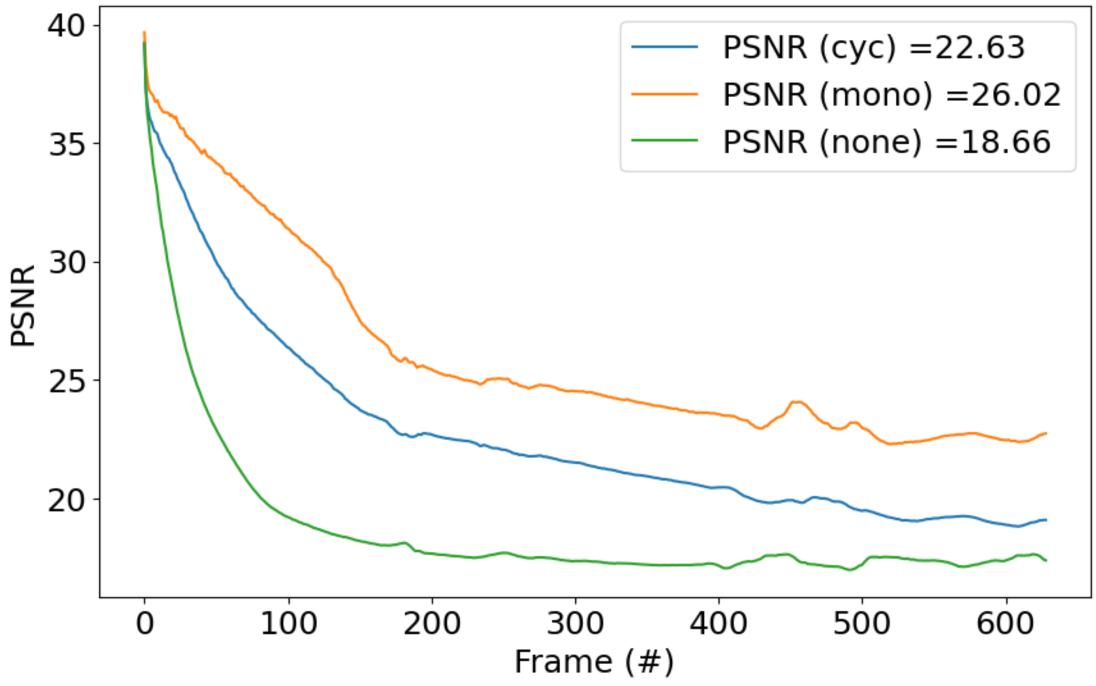

Figure 6. The PSNR-per frame diagram in validation dataset using different strategy

4. Handwrite derivate conditional VAE formula

Conditional VAE derivation

$$P(z,x) = p(x) \times p(z|x)$$

$$L(X,c,q,\theta) = \sum_X \log p(X|c;\theta)$$

$$= \int_z q(z|X,c;\phi) * \log\left(\frac{p(z,x|c;\theta)}{p(z|X,c;\theta)}\right) dz$$

$$= \int_z q(z|X,c;\phi) * \log\left(\frac{P(z,x|c;\theta)}{q(z|X,c;\phi)} \quad \frac{q(z|X,c;\phi)}{p(z|X,c;\theta)}\right) dz$$

$$= \int_z q(z|X,c;\phi) * \log\left(\frac{P(z,x|c;\theta)}{q(z|X,c;\phi)}\right) dz + \underbrace{\int_z q(z|X,c;\phi) * \log\left(\frac{q(z|X,c;\phi)}{\underbrace{p(z|X,c;\theta)}_{\text{intractable}}}\right) dz}_{KL \geq 0}$$

lower bound $\downarrow$

$$= \int_z q(z|X,c;\phi) \times \log\left(\frac{P(X|z,c;\theta) \times p(z|c;\theta)}{q(z|X,c;\phi)}\right) dz$$

$$= \int_z q(z|X,c;\phi) \times \log P(X|z,c;\theta)\, dz + \int_z q(z|X,c;\phi) \times \log \frac{P(z|c;\theta)}{q(z|X,c;\phi)} dz$$

$$= E_{z \sim q(z|X,c;\phi)} \log p(X|z,c;\theta) - KL(q(z|X,c;\phi) \,\|\, \underbrace{p(z|c;\theta)}_{\substack{\text{prior} \\ \downarrow \\ p(z|c)}})$$

5. References

1 E. Denton and R. Fergus, "Stochastic video generation with a learned prior," in *International conference on machine learning*, 2018: PMLR, pp. 1174-1183.

2 H. Fu, C. Li, X. Liu, J. Gao, A. Celikyilmaz, and L. Carin, "Cyclical annealing schedule: A simple approach to mitigating kl vanishing," *arXiv preprint arXiv:1903.10145,* 2019.