Summer 2023 DLP Lab4

312552017 董家鳴

i. Introduction (5%)

In this lab, we need to implement conditional video prediction in a VAE-based model to generate the prediction frame by frame by taking poses and the first image as inputs in inference time. For the training dataset the training time, the generated poses, and images will be provided in this lab. Generally speaking, by predicting frames, we can control the movement of everybody like the woman in figure 1.

Figure 1: control movement of a woman

The requirements in this lab are listed below:

1. **Training details implementation**
   (a) Implement Video prediction protocol in (train) stage
   (b) Implement reparameterization tricks
   (c) KL annealing implementation. (a) Cyclical. (b) Monotonic
2. **Teacher forcing strategy**
   (a) Setup teacher forcing ratio, and plot the diagram in training
   (b) Teacher forcing ratio: $0 \sim 1$
3. **Plot diagrams**
   (a) Plot the loss curve while training
   (b) Plot PSNR-per frame diagram while validation
4. **Analysis**

(a) Compare the result in the loss curve if you apply different KL annealing strategies or even without KL annealing. Which one is better?

5. **Validate your result and make it into gif file**

   (a) Pick one video clips in testing dataset and make the frames generated by your model into a gif file (This should be shown to TAs in LAB4 demo)

6. **Derivation of Conditional VAE (Extra point)**

ii. Implementation Detail (25%)

1. training protocol (10%)

- **Forward pass** I think the coarse version of training in the figure 2 that TA provided has already told the detail of the model's forwarding. By following the flow in the figure 2, we can write down the source code in the figure 3.

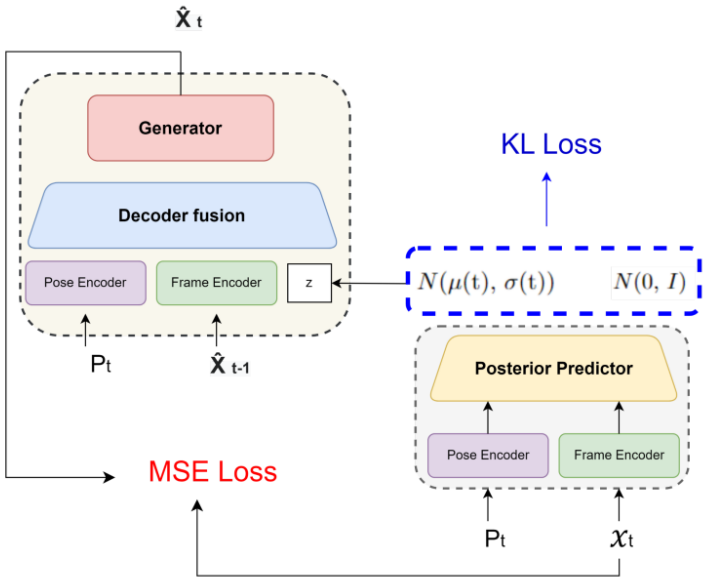

Figure 2: coarse version of the training protocol

```python
def forward(self, p_t, xt_1_hat, x_t):
    pose = self.label_transformation(p_t) # p_t is the current label
    frame_last = self.frame_transformation(xt_1_hat) # xt_1_hat is last prediction, if t-1 == 1, xt_1 is x_1
    frame = self.frame_transformation(x_t) # x_t is the current img


    # encoder
    z, mu, logvar = self.Gaussian_Predictor(frame, pose) # posterior predict

    # decoder
    d = self.Decoder_Fusion(frame_last, pose, z)
    x_t_hat = self.Generator(d)

    # loss
    mse = self.mse_criterion(x_t, x_t_hat)
    kld = kl_criterion(mu, logvar, self.batch_size) # KL divergence


    return x_t_hat, mse, kld
```

Figure 3: predict one future frame

```python
def training_one_step(self, img, label, adapt_TeacherForcing, idx):
    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)

    mse_tol = 0
    kld_tol = 0
    for i in range(1, self.train_vi_len):
        p_t = label[i]

        if i == 1 or adapt_TeacherForcing:
            x_t_1 = img[i - 1]
        else:
            x_t_1  = x_t_hat
        x_t = img[i]


        # generate next frame
        x_t_hat, mse, kld = self.forward(p_t, x_t_1, x_t)

        # loss
        mse_tol += mse
        kld_tol += kld
    return mse_tol, kld_tol
```

Figure 4: one training step

- **Train in one step** Figure 4 shows how I train the model in one step. I first change the tensor into (seq, B, C, H, W) to give different frames for the model. When generating the 16 frames (in this lab). we have no last frame to be taken as Decoder fusion input in the scenario for generating frame x_{t-2}. Hence, x_{t-1} which is img[0] will be provided in the dataset to be the first input of the generative system. Besides, when we're using the teacher-forcing strategy, the last frame will also be the frame provided in the dataset, not the frame generated.

2. reparameterization tricks (5%) VAEs sample from a random node z approximated by the posterior predictor's parametric model $q(Z \mid X, c; \phi)$. Backpropagation cannot flow through a random node since it's not a deterministic function, which means it is not differentiable. So we need to use the "reparameterization trick."

The reparameterization trick works by expressing the random variable z in terms of deterministic variables and a standard normal noise ϵ:

$$z = \mu + \sigma \cdot \epsilon, \quad \epsilon \sim \mathcal{N}(0, 1)$$

As a result of sampling, ϵ is known and fixed at a particular value, so z will become a deterministic function which is differentiable to perform the backpropagation.

3. teacher forcing strategy (5%)

- **Recap RNN** Instead of feeding the actual input x_t to the encoder at each time step, we feed the predicted output from the previous time step. This means that the model is always trying to predict the next frame based on its own predictions, rather than the actual sequence, which is what the RNN does.
- **What is Force** When using teacher forcing, we feed the actual x_t instead of the predicted output from the previous time step. This can help the model to learn more

effectively, as it is being ”forced” to match the actual output, rather than its own predictions.

- **Exposure bias** However, it may force the model to become reliant on having the correct previous output, which it won't have at test time. This is known as the ”exposure bias” problem, and it will lead to poor performance when testing.
- **Teacher forcing** To mitigate this issue, a common strategy is to use teacher forcing with a certain probability, which is often high at the start of training and gradually reduced over time. That is the Teacher forcing ratio (TFR). Here I define some hyperparameters to do that, *tfr_sde* to do the decrease with each epoch, *tfr_d_step* which is a hyperparameter that can control the decay rate at each epoch, and finally, 0 to ensure TFR does not go below 0 since the ratio may be decreased to a negative number in figure 5.
- **My setting** I ran a total of 200 epochs. The concept of teacher forcing is that there must be a teacher to guide at the beginning, so as not to learn randomly. Therefore, there is teacher forcing for the first 20 epochs, but by the time it reaches the 40th epoch, there is no more guidance from the teacher. This is because if the guidance continues, the model's generalization will be poor, and the validation results will be very bad.

```python
def teacher_forcing_ratio_update(self, epoch):
    if epoch >= self.tfr_sde: # start decay to not be forced
        decay_rate = self.tfr_d_step  # set the decay rate in args
        self.tfr = max(self.tfr - decay_rate, 0)  # ensure TFR does not go below 0
```

Figure 5: Teacher forcing

```python
# Teacher Forcing strategy
parser.add_argument('--tfr',          type=float, default=1.0,  help="The initial teacher forcing ratio")
parser.add_argument('--tfr_sde',      type=int,   default=20,   help="The epoch that teacher forcing ratio start to decay")
parser.add_argument('--tfr_d_step',   type=float, default=0.05, help="Decay step that teacher forcing ratio adopted")
```

Figure 6: Teacher forcing setting

4. kl annealing ratio (5%)

- **Posterior collapse** In the early stages of training, the reconstruction loss (encoder part) often dominates the KL divergence term since it just set the mean and variance of q to be as same as p (it is an easy task), leading the VAE to ignore the latent space's structure. This can hinder the model from learning meaningful representations in the latent space.
- **Solution** We can add an increasing constant weight, , to the KL term. is between 0 1. The value of beta will gradually increase as a whole batch of data is processed, meaning that the KL term will be smaller at the beginning. This allows the model to initially focus on training the decoder.
- **Linear increasing** Aka Monotonic increasing. For it, I set the ratio 0.5, which means that starting from the middle, the value of the KL term will not be decreased. It was shown in figure 7.

- **Cyclical increasing** For it, I set the ratio 0.5, which means that starting from the middle of each cycle, the value of the KL term will not be decreased
- **why 0.5** I believe that the middle value is the best choice; extremes are not good.

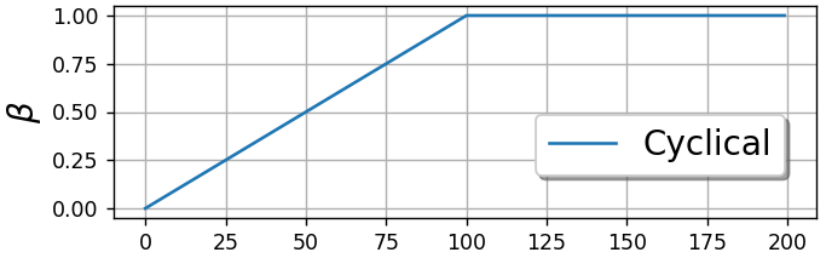

Figure 7: KL annealing - Monotonic

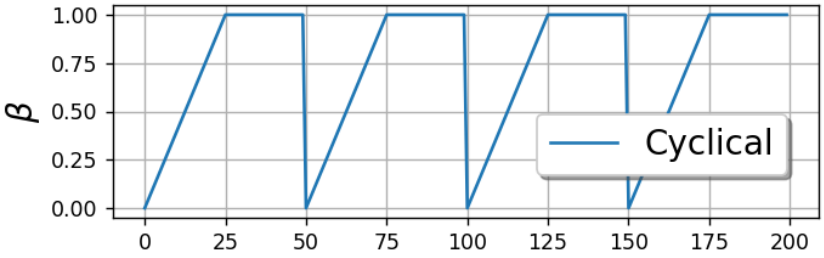

Figure 8: KL annealing - Cyclical

iii. Analysis and Discussion (20% + 5%)

1. Plot Teacher forcing ratio (5%)

- **a. Analysis and comparison with the loss curve** To prevent the model from overfitting, I stopped using ground truth data after the 40th epoch. As seen in Figure 9, the loss oscillates severely between the 20th and 40th epochs. I believe this is because, by using the ground truth as the input for the next time step (as is done with teacher forcing), the model does not learn to recover from its mistakes. This can create a discrepancy between training and inference, leading to higher loss. Sometimes, when teacher forcing is not used and the model has learned something, the loss decreases.

2. loss curve while training (10%) Because the cyclical scheduler constantly changes the value of to modify the size of the KL loss, the loss will oscillate slightly, while the monotonic scheduler will hardly oscillate. This slight oscillation can enable the model to jump to another local minimum, making the model better. This result is consistent with what is described in the paper [1], and the oscillation of the loss can make the model less prone to overfitting the data. Looking at the results, when I trained for 200 epochs, the use of the cyclical method allowed me to reach 26.7 PSNR on the test data, while the monotonic method only reached 23 PSNR because in the later stages of training, the model was no longer able to learn anything new. Originally, I thought that without adding KL annealing, the training would not proceed well. However, the loss still decreased, and after testing, the validation PSNR also reached 23. I am thinking that perhaps the number of training epochs was not sufficient, so the difference between using KL annealing and not using it is not evident.

- **a. With KL annealing (Monotonic)**

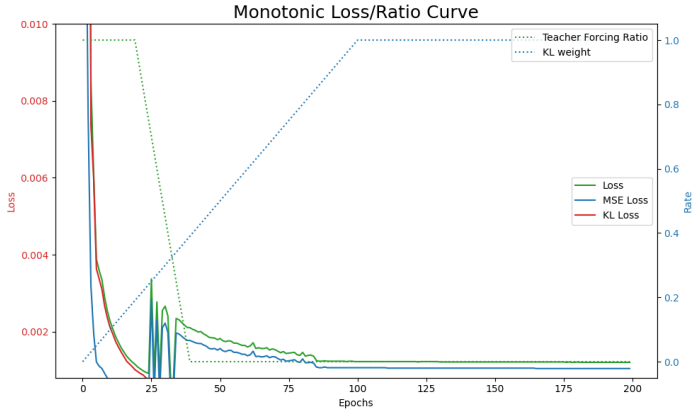

Figure 9: With KL annealing (Monotonic)

- **b. With KL annealing (Cyclical)**

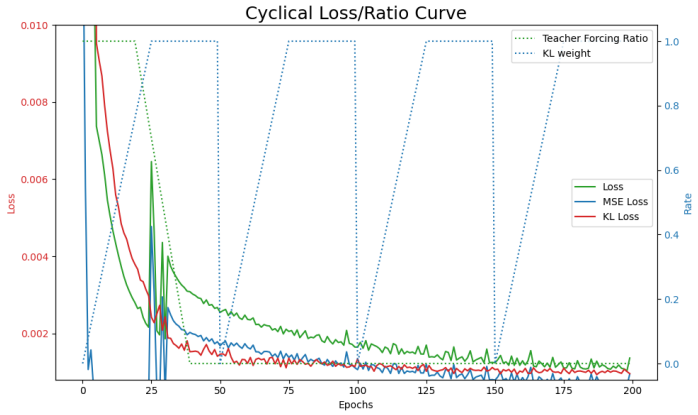

Figure 10: With KL annealing (Cyclical)

- **c. Without KL annealing**

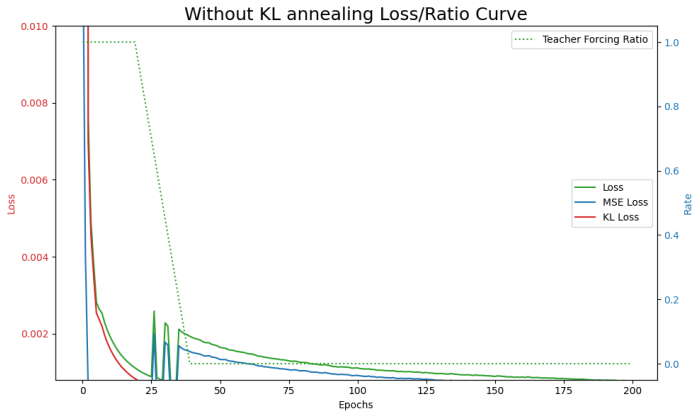

Figure 11: Without KL annealing

3. Plot the PSNR-per frame diagram in validation dataset (5%)

- **Complexity of Data** From the given figure 12, it appears that the PSNR value starts high and then decreases. I think it's because the complexity of data is increasing. If the data is very complex and the model is not capable of capturing that complexity, it may initially perform well on simple patterns but then fail to generalize to more complex structures in the data.

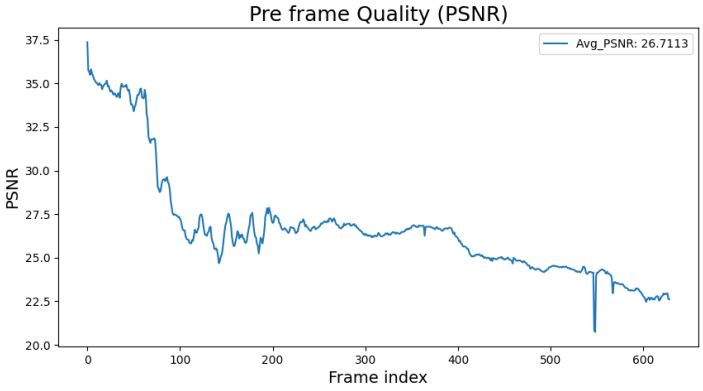

Figure 12: PSNR-per in validation dataset

4. Derivate conditional VAE formula (extra points 5%)

To generate the style of input, CVAE is proposed to use the original data and its condition as the input of the encoder.

CVAE will restrict x to condition c, which is $p(X|c)$. In order to obtain model parameters θ, we must find a way to maximize $p(X|c;\theta)$, which is denoted as follow:

$$p(X|c;\theta) = \int_z p(X|Z,c;\theta)p(Z|c)\,dz$$

Since the equation below is intractable, we will first use log-likelihood to do the maximization. (Note that, the log is a monotone-increasing function, so maximizing the following equation is the same as below.) and we will introduce a new random distribution denoted as $q(z|x, c; \phi)$ to approach $p(X|c; \theta)$.

$$\log p(X|c;\theta) = \int_z q(Z|X,c;\phi) \log p(X|c;\theta)\, dz$$

Next, let's transform it into a more explainable equation.

$$\begin{aligned}
\log p(X|c;\theta) &= \int_z q(Z|X,c;\phi) \log p(X|c;\theta)\, dz \\
&= \int_z q(Z|X,c;\phi) \log \left(\frac{p(Z,X|c;\theta)}{p(Z|X,c;\theta)} \right) dz \\
&= \int_z q(Z|X,c;\phi) \log \left(\frac{p(Z,X|c;\theta)}{q(Z|X,c;\phi)} \right) dz \\
&\quad + \int_z q(Z|X,c;\phi) \log \left(\frac{q(Z|X,c;\phi)}{p(Z|X,c;\theta)} \right) dz \\
&= L_b(X,c,q,\theta) + KL\left(q(Z|X,c;\phi)||p(Z|X,c;\theta)\right)
\end{aligned}$$

where $KL\left(q(Z|X,c;\phi)||p(Z|X,c;\theta)\right)$ is KL divergence of the two distribution, $q(Z|X,c;\phi)$ and $p(Z|X,c;\theta)$. Since KL divergence must ≥ 0, the $L_b(X,c,q,\theta)$ is the evidence lower bound (ELBO).

Thus, we need to maximize the $L_b(X,c,q,\theta)$ in order to maximize $\log p(X|c;\theta)$. Let's transform the equation of $L_b(X,c,q,\theta)$ to connect the relationship between neuron networks.

$$\begin{aligned}
L_b(X,c,q,\theta) &= \int_z q(Z|X,c;\phi) \log \left(\frac{p(Z,X|c;\theta)}{q(Z|X,c;\phi)} \right) dz \\
&= \int_z q(Z|X,c;\phi) \log \left(\frac{p(X|Z,c;\theta)p(Z|c;\theta)}{q(Z|X,c;\phi)} \right) dz \\
&= \int_z q(Z|X,c;\phi) \log p(X|Z,c;\theta)\, dz \\
&\quad + \int_z q(Z|X,c;\phi) \log \left(\frac{p(Z|c;\theta)}{q(Z|X,c;\phi)} \right) dz \\
&= E_{q(Z|X)} \log(p(X|Z,c;\theta)) - KL\left(q(Z|X,c;\phi)||p(Z|c)\right)
\end{aligned}$$

The "KL$(q(Z|X,c;\phi)||p(Z|c))$" is actually the output of the encoder with network parameter ϕ, and $E_{q(Z|X)}\left[\log(p(X|Z,c;\theta))\right]$ is the output of the decoder with network parameter θ. The goal is to minimize the gap between the distribution that X encoded and the Z (decoder), and then maximize the probability of Z that can generate the X (encoder).

References

[1] Hao Fu et al. *Cyclical Annealing Schedule: A Simple Approach to Mitigating KL Vanishing.* 2019. arXiv: `1903.10145 [cs.LG]`.