DLP
Lab4: Conditional VAE for Video Prediction
Report

林睿騰(Rui-Teng Lin)
313551105

April 14, 2025

Contents

1 Introduction

In this lab, we will implement a conditional VAE for video prediction.

1.1 Conditional VAE

The conditional VAE is a type of VAE that takes a data x as input and a conditional data y as input, we want to model the conditional distribution $p(z|x, y)$. In this case, we model the conditional VAE to predict the distribution of the current frame z from the previous frame x and current label y.

To train a conditional VAE, we define a posterior predictor $q(z|x, y)$ that models the approximate posterior distribution of the latent variable z given the observation x and condition y, and a prior predictor $p(z|y)$ that models the prior distribution of z conditioned only on y.

In brief, posterior predictor uses current frame and label to predict the distribution of the latent code, and the distribution of the latent code is modeled to be like a normal distribution, and a decoder fusion to decode the latent code into the current frame.

2 Implementation Details

2.1 Training Protocol

To train the conditional VAE and video prediction model, we need to iterate through the dataset and take the previous frame and current label as input, and then train the model to minimize the loss function.

The detailed training protocol is as follows:

1. Iterate through the dataset and take the previous frame and current label as input.
2. Encode the previous frame, current frame and current label.
3. Use the posterior predictor to predict the distribution of the latent code.
4. Sample the latent code from the distribution.
5. Use the decoder fusion to decode the latent code.
6. Use a generator to generate the current frame.
7. Update the model with MSE loss between the generated frame and the current frame.
8. Update the model with KL divergence loss between the predicted distribution and the normal distribution.

the implementation of the training protocol is as follows:

```
for t in range(1, timesteps):
    if adapt_TeacherForcing: # step 1 - previous frame
        prev = img[:, t - 1, ...]
    else:
        prev = x_hat

    cur = img[:, t, ...] # step 1 - current frame
    cur_label = label[:, t, ...] # step 1 - current label

```

```
    encoded_cur_frame = self.frame_transformation(cur) # step 2 - encoded current frame
    encoded_label = self.label_transformation(cur_label) # step 2 - encoded current label
    encoded_prev_frame = self.frame_transformation(prev).detach() # step 2 - encoded previous frame

    z, mu, logvar = self.Gaussian_Predictor(
        encoded_cur_frame, encoded_label
    ) # step 3, 4 - predict the distribution of the latent code and sample a latent code


    decoded = self.Decoder_Fusion(
        encoded_prev_frame, encoded_label, z
    ) # step 5 - decode the latent code

    x_hat = self.Generator(decoded) # step 6 - generate the current frame
    x_hat = nn.functional.sigmoid(x_hat)

    mse_loss = self.mse_criterion(x_hat, cur) # step 7 - calculate the MSE loss
    kl_loss = kl_criterion(mu, logvar, batch_size) # step 8 - calculate the KL divergence loss

    loss = mse_loss + beta * kl_loss
    total_loss += loss
```

after then, we call the optimizer to update the model parameters.

2.2 Testing Protocol

Testing protocol is more straightforward. We just need to iterate through the dataset and take the previous frame and current label as input, and then generate the current frame.

Detailed protocol is as follows:

1. Iterate through the dataset and take the previous frame and current label as input.
2. Encode the previous frame and current label.
3. Sample a latent code from the normal distribution.
4. Decode the latent code.
5. Generate the current frame.

The implementation of the testing protocol is as follows:

```
for t in range(1, label.shape[0]):
    prev = decoded_frame_list[-1].to(self.args.device) # step 1 - previous frame
    lbl = label[t, ...].to(self.args.device) # step 1 - current label

    encoded_img = self.frame_transformation(prev) # step 2 - encoded previous frame
```

```
    encoded_label = self.label_transformation(lbl) # step 2 - encoded current label

    z, mu, logvar = self.Gaussian_Predictor(encoded_img, encoded_label)
    eps = torch.randn_like(z) # step 3 - sample a latent code

    decoded_frame = self.Decoder_Fusion(encoded_img, encoded_label, eps) # step 4 - decode the latent code

    x_hat = self.Generator(decoded_frame) # step 5 - generate the current frame
    x_hat = nn.functional.sigmoid(x_hat)

    decoded_frame_list.append(x_hat.cpu())
    label_list.append(lbl.cpu())
```

2.3 Reparameterization Trick

To implement the reparameterization trick, we just sample a ϵ from the normal distribution and then use the formula:

$$z = \mu + \sigma \cdot \epsilon \tag{1}$$

to get the latent code.

The implementation of the reparameterization trick is as follows:

```
eps = torch.randn_like(z)
z = mu + torch.exp(logvar / 2) * eps
```

2.4 Teacher Forcing

If a training step is teacher forcing, we just use the groundtruth of previous frame as the input. As for the teacher forcing ratio, we just need to update the ratio in the training protocol when episode reaches the specified epoch tfr_sde, and then decrease the ratio by tfr_d_step every epoch.

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

2.5 KL annealing

KL annealing is a technique to gradually increase the KL divergence loss during training, it can help the model to strive a balance between the reconstruction loss and the KL divergence loss. In this lab, we implemented three types of KL annealing:

1. Cyclical: The KL divergence loss is increased in a cyclic manner.
2. Monotonic: The KL divergence loss is increased monotonically.
3. Constant (equals to disabled): The KL divergence loss is constant.

The following graph shows the difference between the three types of KL annealing.

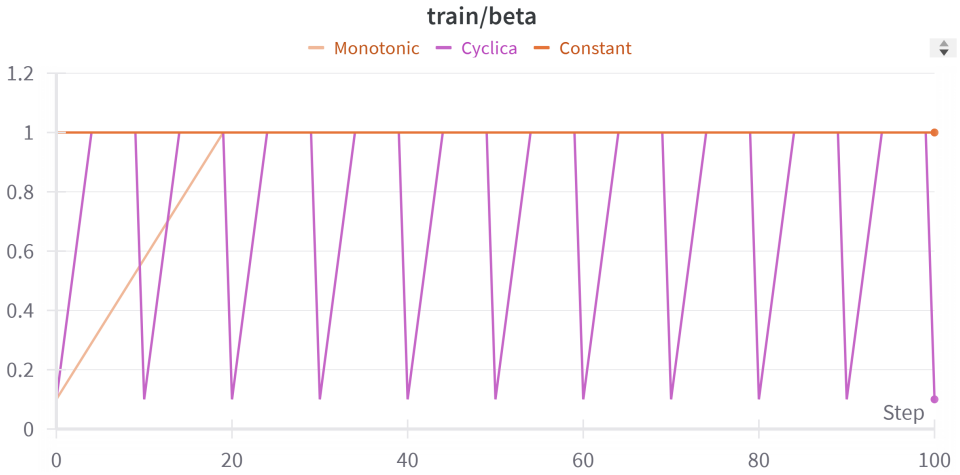


Figure 1: KL annealing

Since I found that using a zero start value would make the model unstable at the beginning, so I set the start value to 0.1, and this prevents all the following experiments to have a NaN loss.

And their implementation is real simple. First, fill all the values with *stop*, and calculate the start of each increasing period by n_iter/n_cycle, then its end is $start + ratio * n\_iter/n\_cycle$. Fill the values between the start and end with $np.linspace(start, stop, int(period * ratio))$, and the rest is the same.

```
betas = np.ones(n_iter) * stop
period = n_iter // n_cycle

for c in range(n_cycle):
    start_ = period * c
    end_ = start_ + int(period * ratio)
    betas[start_:end_] = np.linspace(start, stop, int(period * ratio))

return betas
```

and this is the implementation of the initialization of each type of KL annealing, since for monotic, we can view it as a special case of cyclical with $n\_cycle = 1$:

```
self.betas = self.frange_cycle_linear(
    args.num_epoch,
    n_cycle=args.kl_anneal_cycle,
    ratio=args.kl_anneal_ratio,
) # Cyclical

self.betas = self.frange_cycle_linear(
    args.num_epoch,
    n_cycle=1,
    ratio=args.kl_anneal_ratio,
) # Monotonic

self.betas = np.ones(args.num_epoch) # Constant
```

3 Analysis and Discussion

3.1 Teacher Forcing Ratio

this is the graph of the teacher forcing ratio between two group of runs, which are three runs with different kl annealing, one with decreasing ratio, and one totally disabled TFR:

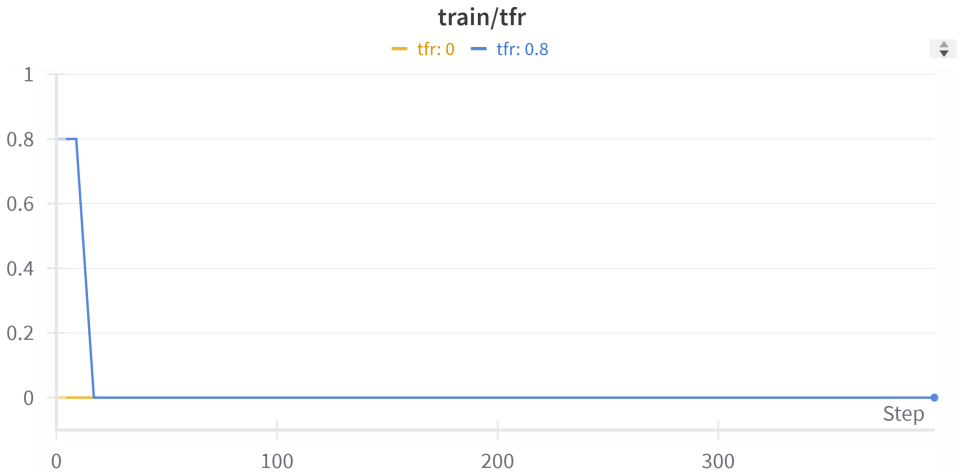


Figure 2: Teacher Forcing Ratio

and the following is the is the training and validation loss graph between the two groups, the solid line is the mean of the group, and the shaded area is the standard deviation:

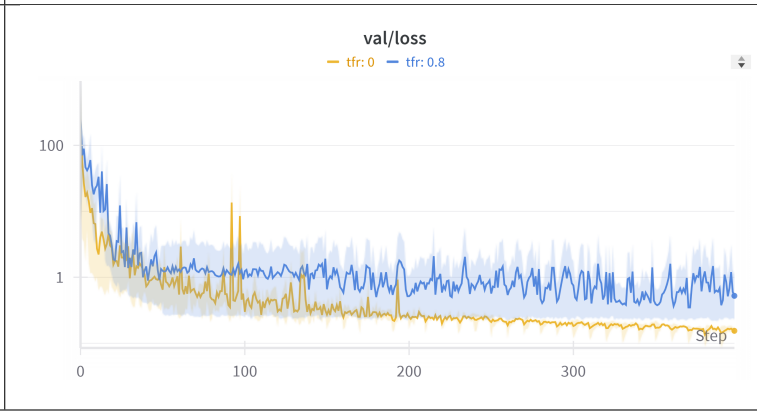
Training Loss	
Validation Loss	

Table 1: Training and Validation Loss

We can see that using TFR does not help the model to converge faster, since it might make the model more sensitive to initial conditions, and converge to a worse local minimum. In contrast, the model with disabled TFR can learn all the things by itself, and converge to a better local minimum.

For the prediction score, the models trained with TFR only reaches 30 to 35, but without TFR, the model can reach 36 to 37.

3.2 KL Annealing

To analyze the effect of different KL annealing strategies, we run three experiments with different KL annealing strategies, and the following is the settings:

Strategy	KL Cycle	KL Ratio
Cyclical	40	0.5
Monotonic	1	0.5
Constant	-	-

Table 2: KL Annealing Settings

And the following table shows the betas, training loss, and validation loss of the three strategies:

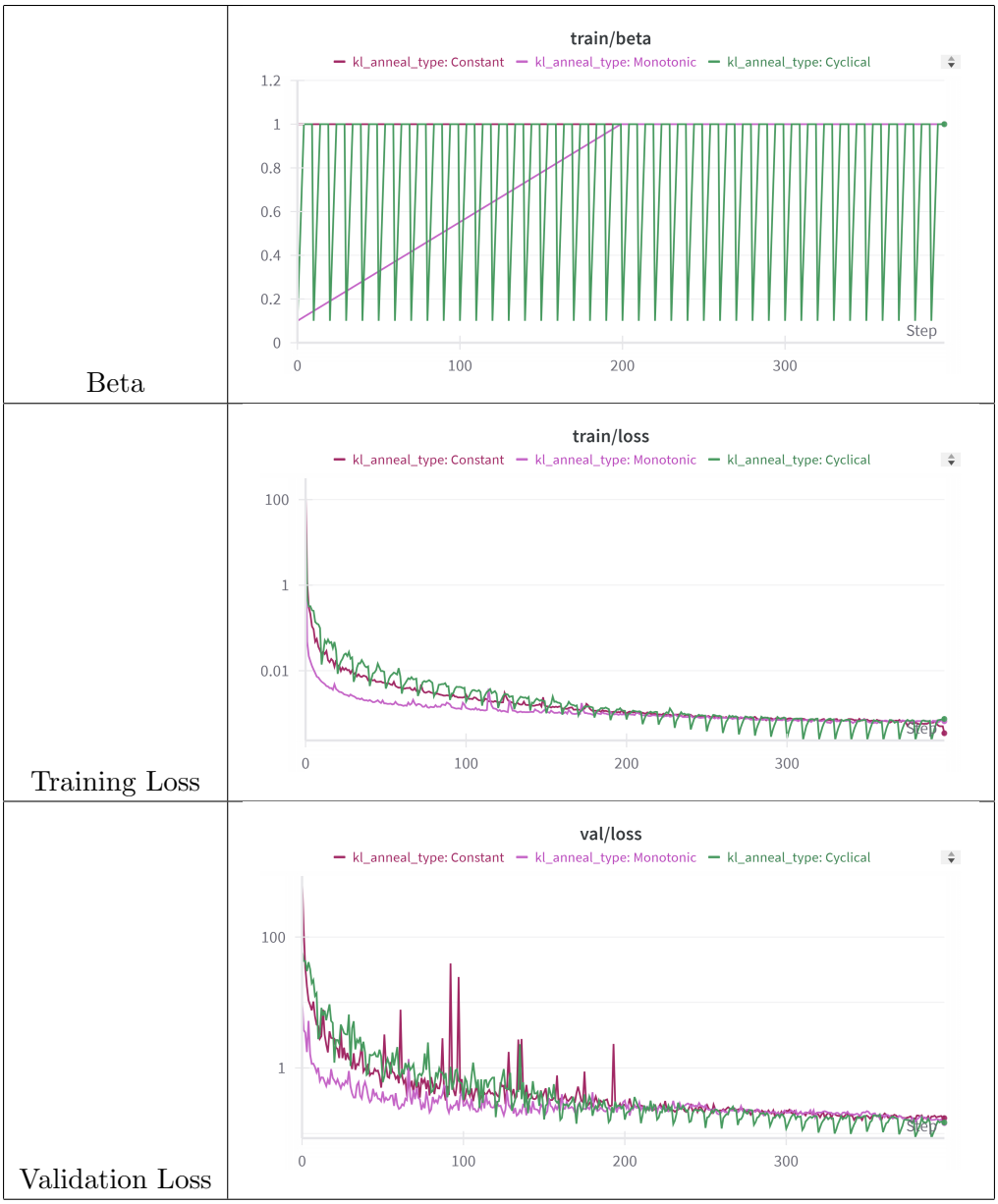


Figure 3: KL Annealing Analysis

We can see from the graph, monotonic KL annealing converges the fastest by its low limitation on the KL divergence loss, and finally reaches a validation loss about 0.14, which is about the same as other two strategies. Constant KL annealing converges slower than monotonic because of its high limitation on the KL divergence loss, but it converges steadly because its target is always the same. Cyclical KL annealing converges the slowest, and the most unstable at the beginning, since its target loss changes in a cyclic manner, so it oscillates in the beginning. But in later stage when all of the three strategies have converged, it can reach a lower loss by its scheduling of the target loss to strike a balance between the reconstruction loss and the KL divergence loss.

3.3 PSNR per frame

the following is the PSNR per frame of the final model, which is trained with parameters in 3:

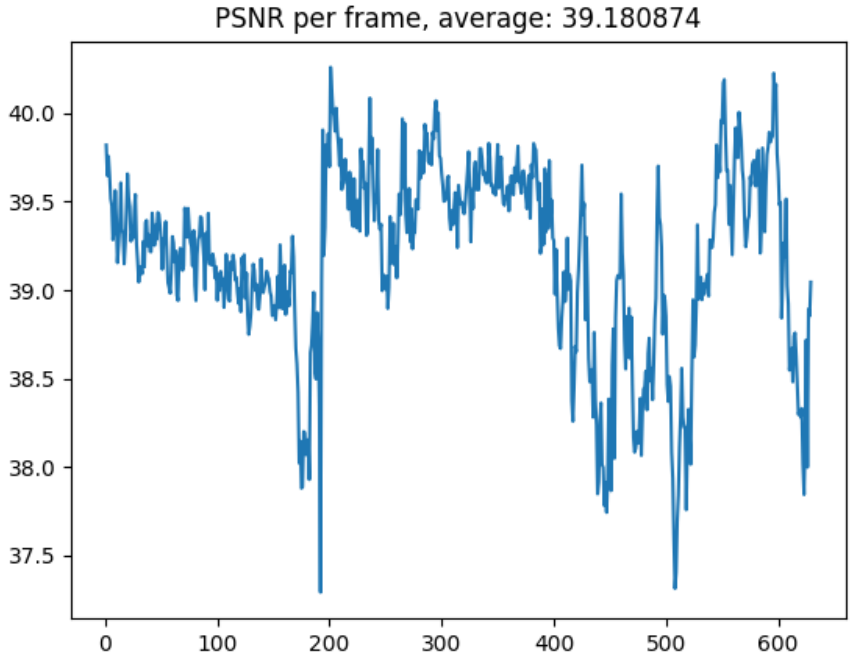


Figure 4: PSNR per frame

We can see that it performs well in most of the frames, with average psnr about 39, and its lowest is about 37.

3.4 Other training strategy analysis

At first, I tried some combinations of parameters with a long epoch, but they all have a score around 33, so I was thinking how to improve the performance. After some experiments, I observed that the images are normalized to be in the range of $[0, 1]$, but the frame generator model output is not normalized with sigmoid function, so I added a sigmoid function to the model output.

After the modification, the model can reach a score around 36, some of them even reach 37.

A Appendix

A.1 Training Parameters of the Final Model

Parameter	Value
Fast training	False
Learning Rate	$3e-4$
Optimizer	AdamW
Scheduler	CosineAnnealingLR
Batch Size	2
KL Annealing Type	Cyclical
KL Annealing Cycle	40
KL Annealing Ratio	0.5
Teacher Forcing Ratio	0.0
Number of Epochs	400

Table 3: Training Parameters of the Final Model