Lab 4 Report

Deep Learning

Name: Kai-Jie Lin
Student ID 110652019

May 4, 2024

1 Derivate conditional VAE formula

From original VAE objective, we want to maximize the maximum log likelihood of the data x:

$$\log p(x;\theta) = KL(q(z|x;\phi)||p(z|x;\theta)) + \int q(z|x;\phi) \log \frac{p(x|z;\theta)}{q(z|x;\phi)} dz$$

Where decoder p is parameterized by θ and encoder q is parameterized by ϕ.
Consider the conditional VAE, we want to maximize the maximum log likelihood of the data x and the conditional variable c:

$$\begin{aligned} \log p(x,c;\theta) &= KL(q(z|x,c;\phi)||p(z|x,c;\theta)) + \int q(z|x,c;\phi) \log \frac{p(x|z,c;\theta)}{q(z|x,c;\phi)} dz \\ &\geq \int q(z|x,c;\phi) \log \frac{p(x|z,c;\theta)p(c|x;\theta)}{q(z|x,c;\phi)} dz \end{aligned}$$

We want to maximize the variational lower bound of the log likelihood. So we can derive the loss function:

$$\begin{aligned} L(x,c,q,\theta) &= \int q(z|x,c;\phi) \log \frac{p(x|z,c;\theta)p(c|x;\theta)}{q(z|x,c;\phi)} dz \\ &= \int q(z|x,c;\phi) \log \frac{p(x|c;\theta)p(z|c;\theta)}{q(z|x,c;\phi)} dz \\ &= \int q(z|x,c;\phi) \log \frac{p(x|c;\theta)}{q(z|x,c;\phi)} dz + \int q(z|x,c;\phi) \log p(x|z,c;\theta) dz \\ &= -KL(q(z|x,c;\phi)||p(z|c;\theta)) + E_{z \quad q(z|x,c;\phi)}[\log p(x|z,c;\theta)] \end{aligned}$$

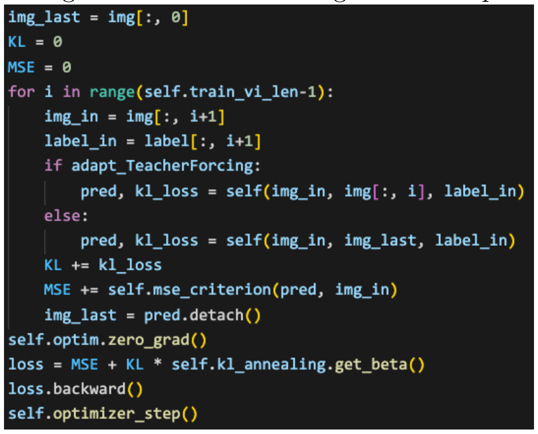
2 Introduction

In this lab, I implemented conditional VAE to do video prediction. The model is trained on 23k video frames and tested on 5 video seqeunces with 630 frames. I experimented various training strategies and hyperparameters to improve the performance of the model. The details of the implementation and the analysis of the results are presented in the following sections.

3 Implementation details

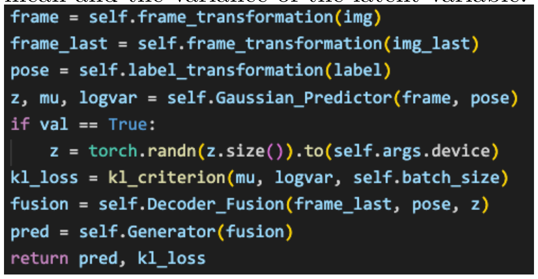
3.1 Training protocol

Training Stage: In one traning epoch, I first determine whether adapting teacher forcing strategy or not. Then, sample a batch of video frames from the training dataset. In one traning step, there will be 16 frames in a batch. I made 15 times forward pass and calculate the mean squared error between the predicted frame and the ground truth frame. If I adapt teacher forcing strategy, I will use the ground truth frame as the input of the next time step. Otherwise, I will use the predicted frame as the input of the next time step. I also calculate the kl divergence between the latent variable and the prior distribution. Finally, I sum the loss of the mean squared error and the kl divergence times kl annealing beta and update the model.



```
img_last = img[:, 0]
KL = 0
MSE = 0
for i in range(self.train_vi_len-1):
    img_in = img[:, i+1]
    label_in = label[:, i+1]
    if adapt_TeacherForcing:
        pred, kl_loss = self(img_in, img[:, i], label_in)
    else:
        pred, kl_loss = self(img_in, img_last, label_in)
    KL += kl_loss
    MSE += self.mse_criterion(pred, img_in)
    img_last = pred.detach()
self.optim.zero_grad()
loss = MSE + KL * self.kl_annealing.get_beta()
loss.backward()
self.optimizer_step()
```

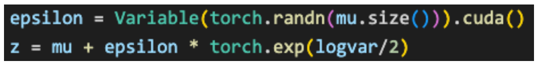
Model Forward: In one forward pass, I first input the current frame and last frame to the frame encoder. Then, input the label into the label encoder. We can predict the latent variable, the mean and the variance of the latent variable from gaussian predictor that takes the output of the frame encoder and the label encoder as input. Finally, we input last frame, current label and the latent variable to the frame decoder to predict the next frame. KL divergence can be calculated by the mean and the variance of the latent variable.



```
frame = self.frame_transformation(img)
frame_last = self.frame_transformation(img_last)
pose = self.label_transformation(label)
z, mu, logvar = self.Gaussian_Predictor(frame, pose)
if val == True:
    z = torch.randn(z.size()).to(self.args.device)
kl_loss = kl_criterion(mu, logvar, self.batch_size)
fusion = self.Decoder_Fusion(frame_last, pose, z)
pred = self.Generator(fusion)
return pred, kl_loss
```

3.2 Reparameterization tricks

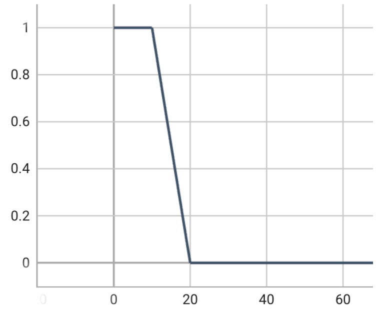
We want to sample the latent variable from the gaussian distribution. But we can't backpropagate through the sampling operation. So we use the reparameterization trick to sample the latent variable. $z = \mu + \epsilon\sigma$, where z is the latent variable, mu is mean of z, σ is standard deviation of z, ϵ is sampled from the standard normal distribution.



```
epsilon = Variable(torch.randn(mu.size())).cuda()
z = mu + epsilon * torch.exp(logvar/2)
```

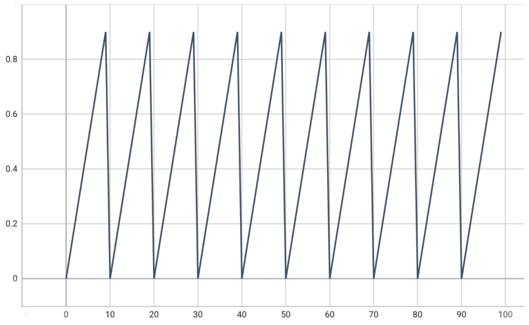
3.3 Teacher forcing strategy

There are three parameters in the teacher forcing strategy: initial teacher forcing ratio, epoch that start to decay, and decay step. If the current epoch is less than the epoch that start to decay, the teacher forcing ratio will be the initial teacher forcing ratio. Otherwise, the teacher forcing ratio decrease linearly.
For example, if the initial teacher forcing ratio is 1.0, the epoch that start to decay is 10, and the decay step is 0.1. We can plot the teacher forcing ratio with respect to the epoch.

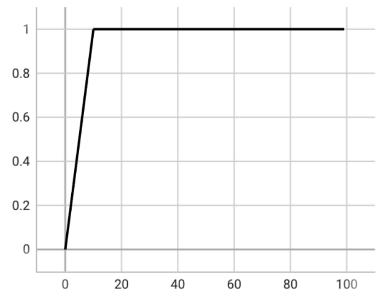


3.4 Kl annealing ratio

There are three types of kl annealing: Cyclical, Monotonic and Non-annealing.
Cyclical:
β is the kl annealing ratio, T is number of total epoches, t is current epoch, M is kl annealing cycle and R is kl annealing ratio.
With $\tau = \frac{\mod(t, \lceil T/M \rceil)}{T/M}$. If $\tau \leq R$, $\beta = \tau/R$, otherwise $\beta = 1$. For example, if $T = 100$, $M = 10$, $R = 1.0$, we can plot the kl annealing ratio with respect to the epoch.



Monotonic:
$\beta = \min(1, tM/RT)$
For example, if $T = 100$, $M = 10$, $R = 1.0$, we can plot the kl annealing ratio with respect to the epoch.

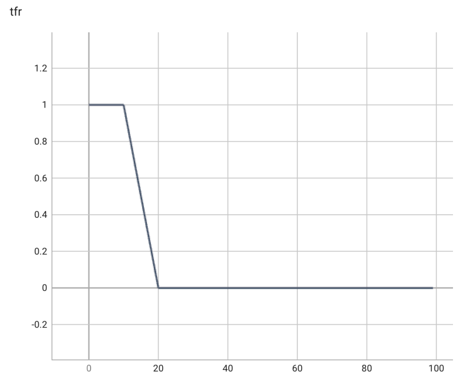


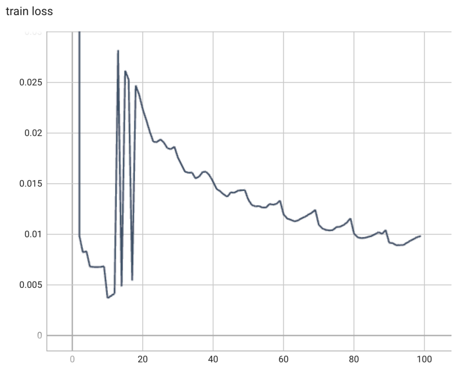
Non-annealing:
$\beta = 1.0$

4 Analysis & Discussion

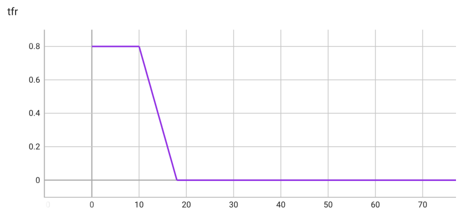
4.1 Teacher forcing ratio

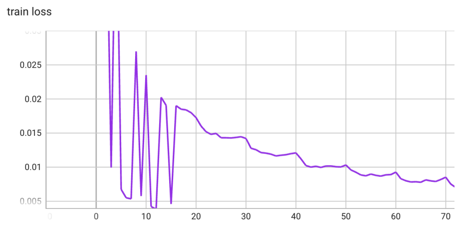
All experiments below are trained with kl annealing ratio = 1.0, kl annealing type = Cyclical, kl annealing cycle = 10 and Adam optimizer.
Case1: tfr = 1.0, tfr_sde=10, tfr_d_step=0.1





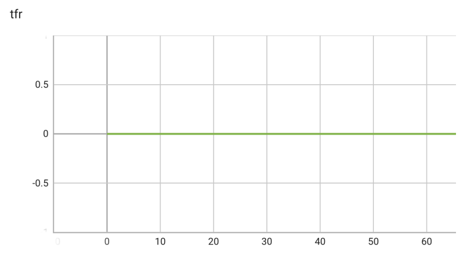
We can observe that the loss first decrease and then increase after 10 epoches. The model would first learn the distribution from the ground truth instead of the predicted frame. After 10 epoches, the model would learn the distribution from the predicted frame. This is not efficient for model to learn conditional generation from the predicted frames.
Case2: tfr = 0.8, tfr_sde=10, tfr_d_step=0.1

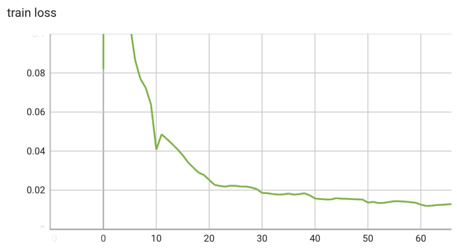




We can observe that the loss is not stable at the first 10 epoch. That is because the model have some probability to learn the distribution from the predicted frame. The loss is lower than the same epoch of case1 (ex. 40 epoch).

Case3: tfr = 0.0, tfr_sde=10, tfr_d_step=0.1

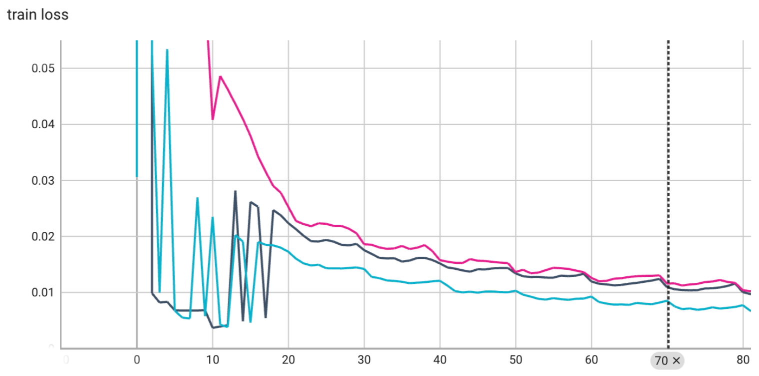




The loss decrease more smoothly than the previous two cases. The model learn the distribution directly from the predicted frame from the beginning. However, the loss is higher than the previous two cases.

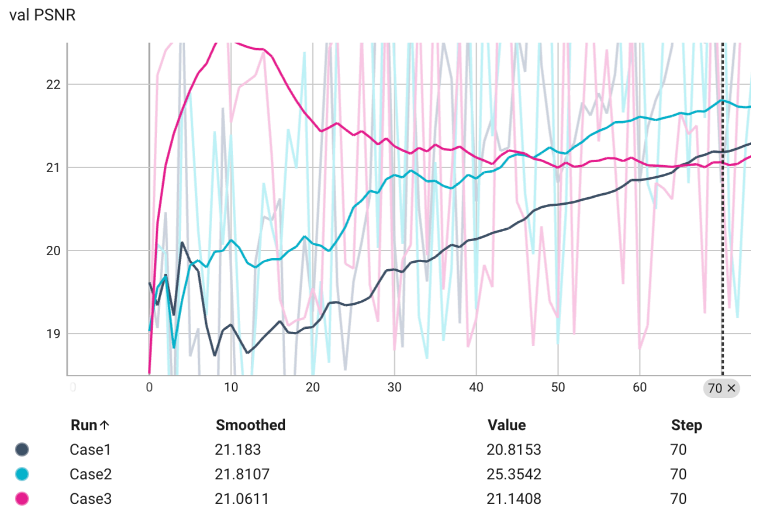
Compare the three cases:

Loss:



	Run↑	Value	Step	Relative
●	Case1	0.0109	70	3.346 hr
●	Case2	0.0085	70	2.248 hr
●	Case3	0.0116	70	2.072 hr

Validation PSNR:

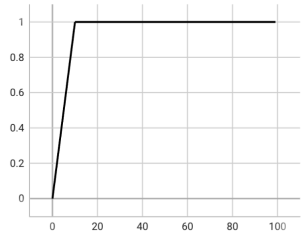
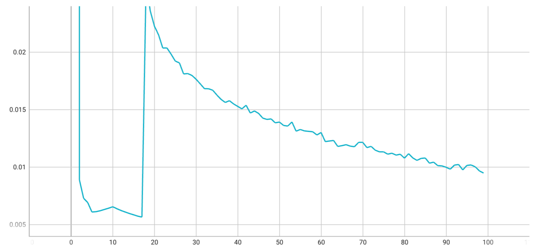


	Run↑	Smoothed	Value	Step
●	Case1	21.183	20.8153	70
●	Case2	21.8107	25.3542	70
●	Case3	21.0611	21.1408	70

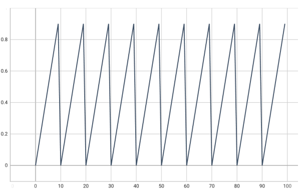
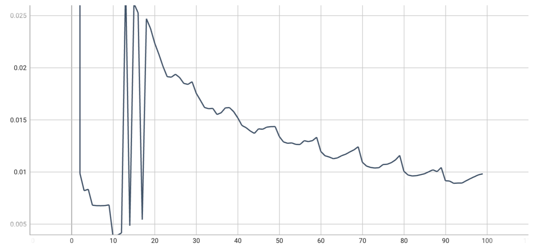
Conclusion:
We can see that the model with tfr $= 0.8$ have the lowest loss and the highest validation PSNR. Lower tfr can make model learn faster and perform better.

4.2 Plot the loss curve while training with different settings.

All experiments below are trained with tfr $= 1.0$, tfr_sde=10, tfr_d_step=0.1 and Adam optimizer
With KL annealing (Monotonic):
$\beta = \min(1, tM/RT)$, $T = 100$, $M = 10$, $R = 1.0$.
Left: β / Right: Loss.

The loss decrease smoothly after 20 epoches.
With KL annealing (Cyclical):
$\tau = \frac{\bmod(t, \lceil T/M \rceil)}{T/M}$. If $\tau \leq R$, $\beta = \tau/R$, otherwise $\beta = 1$.
$T = 100$, $M = 10$, $R = 1.0$.
Left: β / Right: Loss.

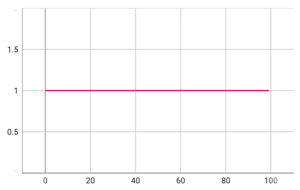
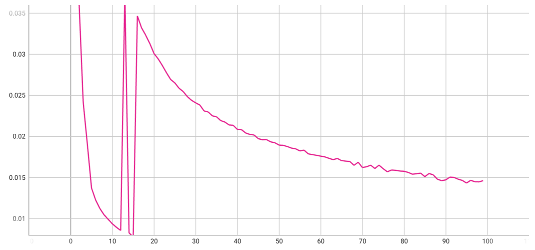



We can observe that the loss would decrease and increase periodically since the cyclical kl annealing.

With KL annealing (None):

$\beta = 1.0$.

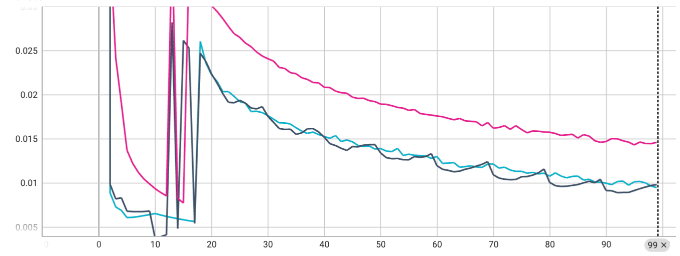
Left: β / Right: Loss.

The loss curve is quite similar to the monotonic kl annealing but the value is larger.

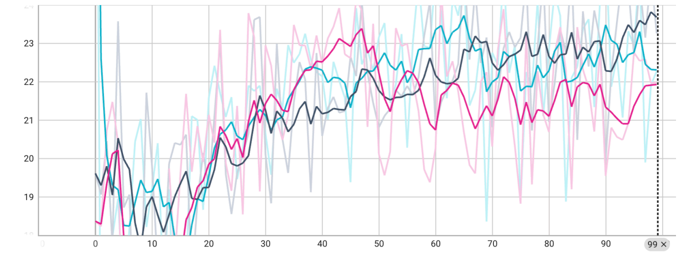
Compare three settings:

Loss:



	Run↑	Value	Step	Relative
●	Cyclical	0.0098	99	4.737 hr
●	Monotonic	0.0095	99	4.739 hr
●	None	0.0146	99	4.73 hr

Validation PSNR:



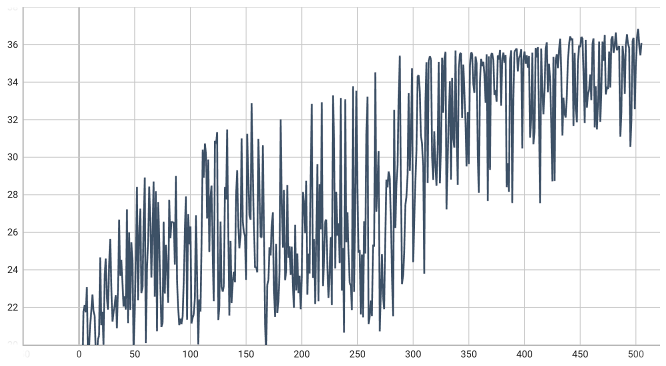
	Run↑	Smoothed	Value	Step	Relative
●	Cyclical	23.6813	23.2626	99	4.737 hr
●	Monotonic	22.3105	22.2787	99	4.739 hr
●	None	21.9312	21.9823	99	4.73 hr

From the comparison plots above, we can conclude that kl annealing indeed helps model training.

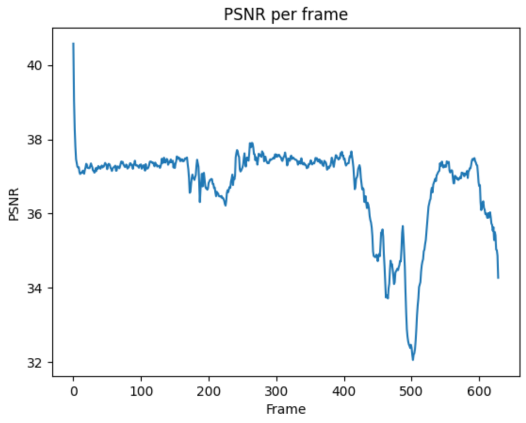
The model with cyclical kl annealing have the lowest loss and the highest validation PSNR.

4.3 Plot the PSNR-per frame diagram in validation dataset

From the experiments above, we can see that the model with tfr = 0.8, ,tfr_sde=10, tfr_d_step=0.1, kl annealing type = Cyclical, kl annealing cycle = 50, kl annealing ratio = 1.0 and Adam optimizer may have the best performance.
Here is the validation PSNR of each epoch after training for 500 epoches:



And the PSNR-per frame diagram tested by 500 epoch model:

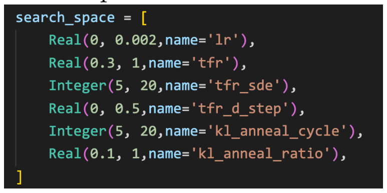


We can observe that the PSNR value drop dramatically at around 500 frames. I guess that is because the person we want to predict have big movement in the video.

4.4 Other training strategy analysis

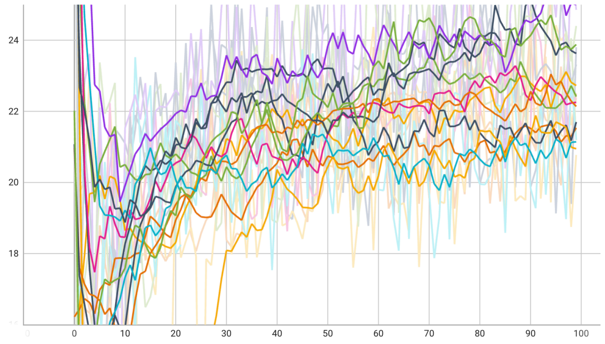
Use SGD optimizer to converge the performance:
I use SGD optimizer to reach higher PSNR value after training for 500 epoches using Adam optimizer. Since SGD directly update the model parameters, it may be more efficient to reach the optimal we want. This helps me get higher score on Kaggle.
Bayesian Optimization:

I use Bayesian Optimization to search the best hyperparameters for the model. Bayesian optimization is a sequential design strategy for global optimization of black-box functions that does not assume any functional forms. It is usually employed to optimize expensive-to-evaluate functions.
Search space:



```
search_space = [
    Real(0, 0.002,name='lr'),
    Real(0.3, 1,name='tfr'),
    Integer(5, 20,name='tfr_sde'),
    Real(0, 0.5,name='tfr_d_step'),
    Integer(5, 20,name='kl_anneal_cycle'),
    Real(0.1, 1,name='kl_anneal_ratio'),
]
```

From the result of Bayesian Optimization, I found that tfr around 0.3 to 0.8 may have better performance.


24
22
20
18
0
10
20
30
40
50
60
70
80
90
100