

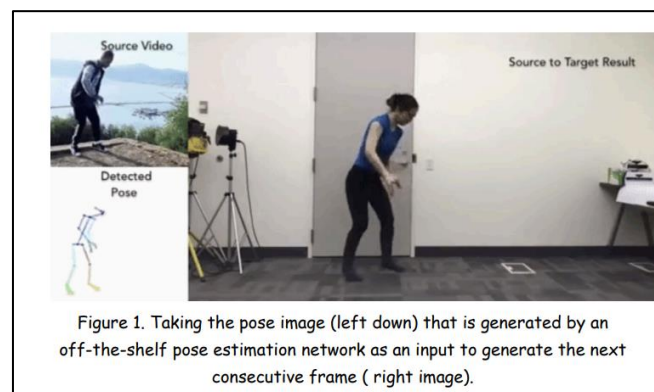
Lab4 – Conditional VAE for Video Prediction

312554006 羅名志

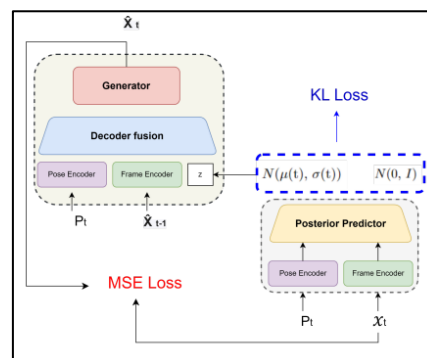
// The deviation of conditional VAE formula is in Part 6

1. Introduction of Lab 4

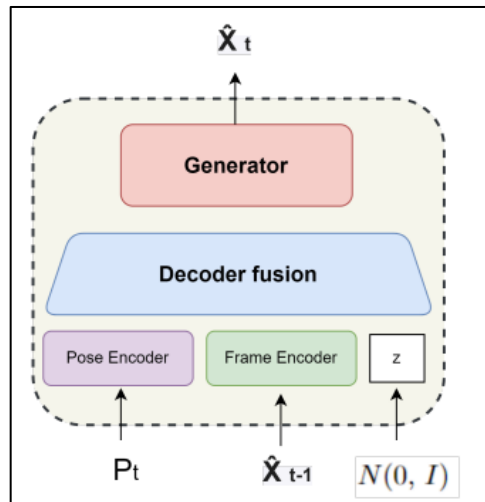
This lab focuses on implementing conditional video prediction in a VAE-based model. Specifically, the task involves predicting the next video sequence given a label (the pose of a person) and the previous video sequence.



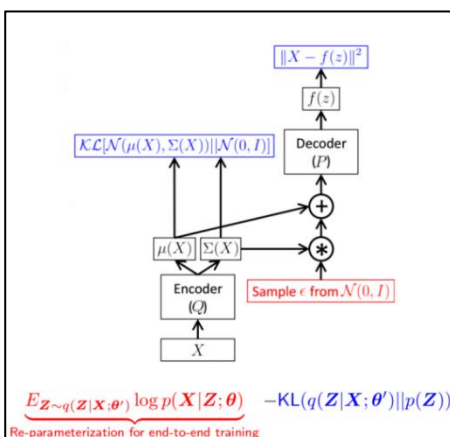
As illustrated in the figure below, during the training stage, the model consists of a posterior predictor, which takes the current frame and the current pose label to obtain a noise sample. Meanwhile, the generator utilizes the current label and the previous sequence along with noise sampled from the posterior predictor to generate the next sequence.



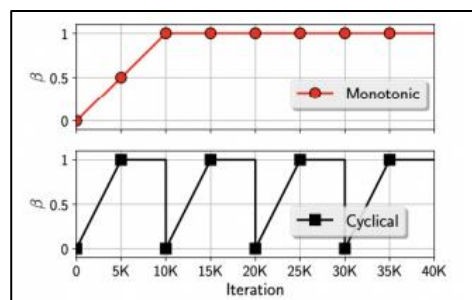
During inference, we solely employ the trained generator to predict future video frames, as depicted in the graph below.



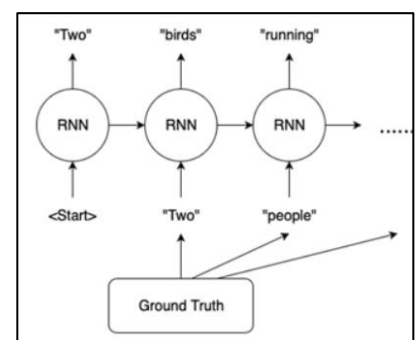
The dataset comprises 23,410 training images, 630 validation images, and 5 testing video sequences, each with 630 labeled frames. This lab also encompasses various training strategies, including KL annealing, teacher forcing, and the reparameterization trick.



Reparameterization trick



KL annealing strategy



Teacher forcing

2. Implementation details

(1) How do you write your training protocol

First, permute the images and labels to the size of (seq, Batch size, Channels, Height, Width). Since we need to use the previous sequence to predict the future sequence, by doing this, we can traverse the following frame in the correct order.

Then, I drop the forward function; instead, I write the whole block in the training_one_step function. I first encode all the labels by label_transformation to H_label and images (frames) by frame_transformation to H_frame. Meanwhile, I append the first transferred frame to H_pred. This can help when predicting the next frame when not using teacher forcing. Each inference output will be appended to this list for future decoding.

```
# TODO
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)
# print(img[0].size()) # ([2,3,32,64]: [B, C, H, W])
# first extract all
H_frame = []
H_label = []
for i in range(self.train_vi_len):
    H_frame.append(self.frame_transformation(img[i]))
    H_label.append(self.label_transformation(label[i]))
H_pred = []
H_pred.append(H_frame[0])
```

Following the model architecture provided in this lab, I first input the transferred frame and transferred label into the Gaussian_Predictor to obtain the noise distribution. With the noise and the previous output stored in H_pred, along with the true label H_label, the Decoder_Fusion and Generator can generate the next frame. After obtaining the next sequence output frame, I calculate the loss using mean squared error and KL divergence. Finally, I backward the loss and update the model weights.

```

recon_loss = 0
kl_loss = 0
for i in range(1,self.train_vi_len):
    z, mu, logvar = self.Gaussian_Predictor(H_frame[i], H_label[i])
    if adapt_TeacherForcing==True:
        fusion = self.Decoder_Fusion(H_frame[i-1], H_label[i], z)
    else:
        fusion = self.Decoder_Fusion(H_pred[i-1], H_label[i], z)
    output = self.Generator(fusion)
    output_trans = self.frame_transformation(output)
    H_pred.append(output_trans)

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

beta = self.kl_annealing.get_beta()

loss = recon_loss + beta*kl_loss
self.optim.zero_grad()
loss.backward()
self.optim.step()

return loss

```

Two things to clarify: First, the loop starts from the second frame and label because the first frame is used as the starting point for prediction. Also, when using teacher forcing, there's no need to use the previous output frames to predict the next one; instead, we can directly use the ground truth image for prediction.

(2) How do you implement reparameterization tricks

As explained in the lab specification, adopting a reparameterization trick is useful for stable training. When dealing with log variance, I first convert the log variance to standard deviation and then randomly sample from a normal distribution, using the formula: $z = \mu + \exp * \text{std}$.

By doing this, we can ensure that the model can be trained end-to-end because the latent space becomes a deterministic function of a noise variable, where the noise won't be backpropagated. This ensures the model remains differentiable.

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

(3) How do you set your teacher forcing strategy

The method of teacher forcing strategy I employ is the simplest case, where I first define the teacher forcing ratio and decrease it. This may ensure that the entire model performs better in the early stages with the assistance of teacher forcing and, in the later epochs, tries to infer better outputs independently. To elaborate, the update of the teacher forcing ratio will first identify the start of

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

the ratio decay, denoted as `tfr_sde`. If the current epoch is smaller than `tfr_sde`, then the model will continue to use teacher forcing. However, after `tfr_sde`, the ratio will decrease linearly each epoch by a step size of `tfr_d_step`, while the model generates a random number as depicted in the first graph. If the random number is smaller than the ratio, the model will continue to use teacher forcing. This process continues until the ratio reaches 0, indicating that the teacher forcing ratio is no longer used, and the model will totally infer the outputs based on its previous outputs.

(4) How do you set your kl annealing ratio

Using KL annealing may improve the model performance and stability by balancing the trade-off between two conflicting objectives during training: reconstruction accuracy and the adherence to the desired latent space distribution. To be more specific, when training VAE, the model may ignore the input data and focuses solely on matching the prior distribution, leading to posterior collapse problem. KL annealing may be a good strategy to address the issue.

There are 3 kinds of kl annealing strategy I use in this Lab.

a. with cyclical updating the Beta

```
elif self.mode == "Cyclical":
    if self.current_epoch%self.cycle_epoch < self.thres_cyclical:
        self.weight = 1/(self.thres_cyclical)* (self.current_epoch%self.cycle_epoch)
    else:
        self.weight = 1
```

The term "thres_cyclical" represents the epoch at which the beta value should start to keep at 1. When the current epoch % each cycle epoch is smaller than the threshold, beta will increment from 0 to 1 epoch by epoch.

b. with monotonic

```
if self.mode == 'Monotonic':
    if self.current_epoch < self.thres_monotonic:
        self.weight = 1/(self.thres_monotonic)*self.current_epoch
    else:
        self.weight = 1
```

I set the thres_monotonic to be $0.25 \times \text{total epochs}$. In other words, the first quarter of epochs' beta value will increment from 0 to 1. Theoretically, this will address the posterior collapse problem by increasingly increment the value of beta at early stages.

c. without any strategy

```
else:
    self.weight = 1
```

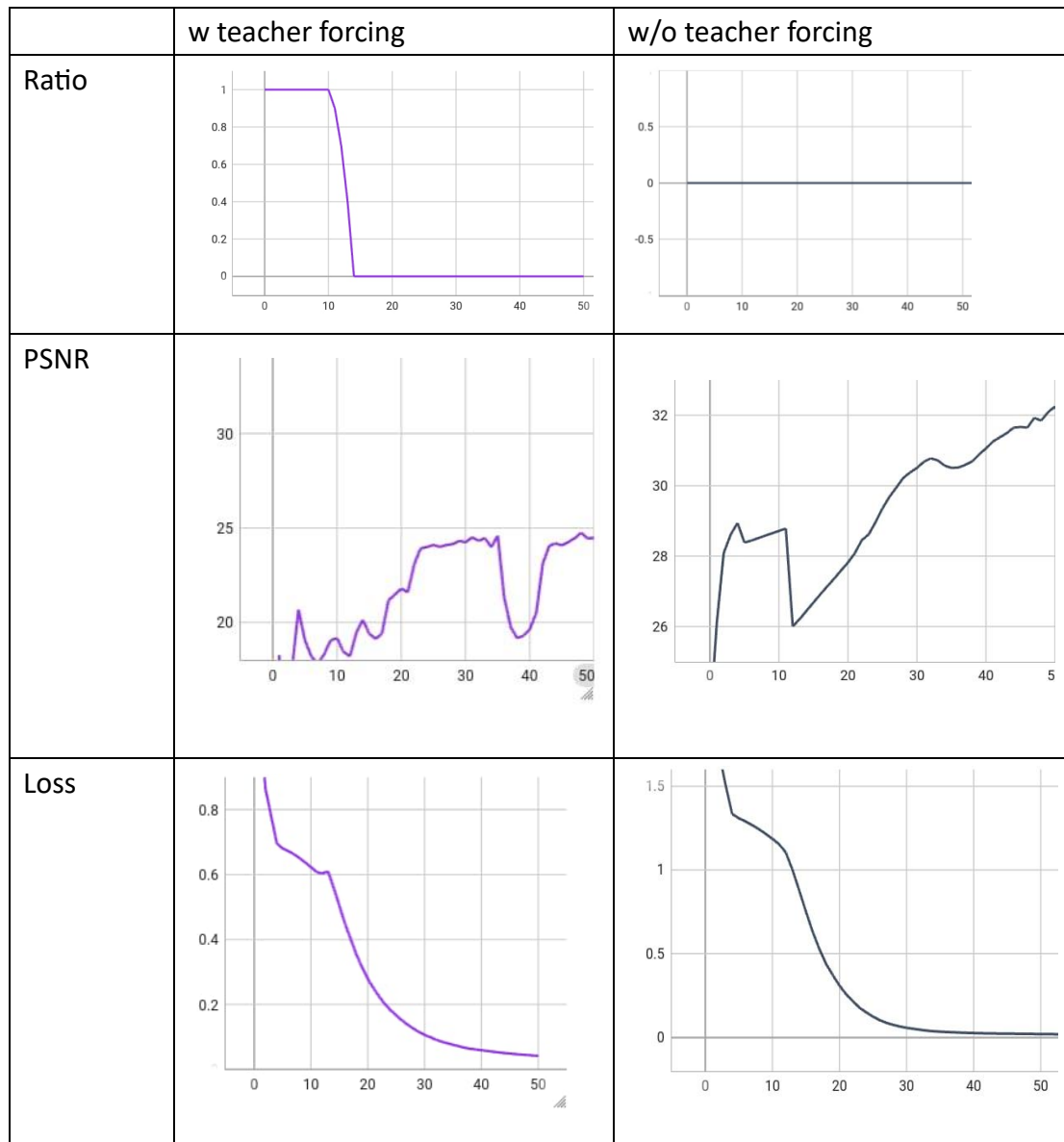
Directly set the beta value to be 1.

3. Analysis and Discussion

(1) Plot teacher forcing ratio

a. Analysis & compare with the loss curve

// No any KL annealing strategy used in this scenario; x-axis is number of epochs.



Analysis:

a. As demonstrated in the comparison above, both PSNR and Loss metrics indicate that employing teacher forcing may result in inferior outcomes. This discrepancy likely arises from the disparity between training with teacher forcing and inference without it. Another contributing factor could be exposure bias, where the model

predominantly learns from straightforward cases when ground truths are provided, potentially leading to instability in generating outputs during the inference stage.

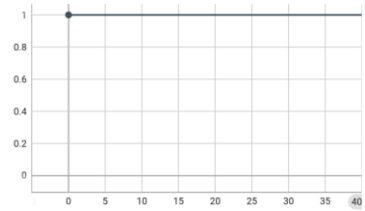
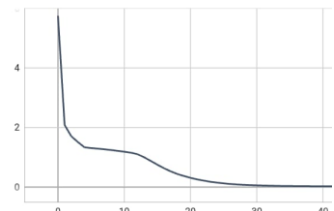
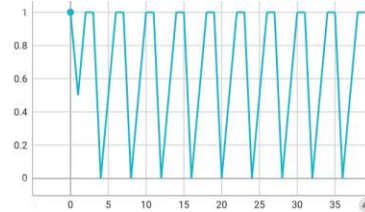
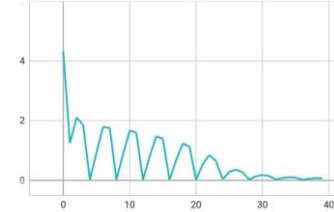

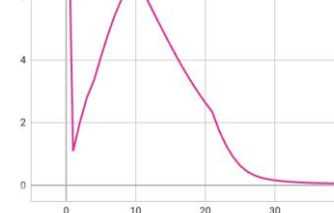
b. The PSNR curve of the model utilizing teacher forcing exhibits more fluctuations compared to the one trained without it. This variation is likely attributed to the latter model being trained from scratch and commencing its learning process independently, without relying on any labels. Consequently, it achieves better and more consistent performance.

c. As depicted in the graph, when teacher forcing begins to decrease (epoch=10), there is a slight drop in the model's PSNR and a minor increase in Loss. This can be attributed to the model transitioning to learning outputs without relying on the ground truth labels, which can initially affect performance metrics.

(2) Plot the loss curve while training with different settings. Analyze the difference between them.

// Since the model without teacher forcing has better performance, here I only show the model without using teacher forcing.

//The average PSNR score refers to the public score obtained in the Kaggle competition; x-axis is the number of epochs.

	Beta distribution	Loss	PSNR
w/o KL annealing			30.4173
w Cyclical			27.6458
w Monotonic			29.0215

Analysis:

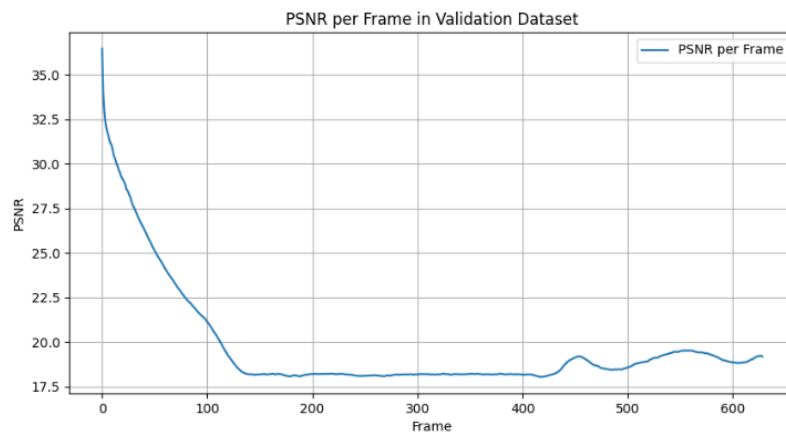
- a. Since cyclical KL annealing updates the beta value periodically, it leads to fluctuations in the loss function.
- b. The loss dramatically spikes during monotonic annealing when epoch=10, possibly due to the unstable values of beta, causing irregular performance in the loss function.
- c. Monotonic KL annealing typically starts with a low beta value of 0 and gradually increases it to 1, theoretically addressing the so-called collapse problem. However, in this case, there is no collapse problem. As evidenced by the relatively smooth curve of the original model without any KL annealing, it suggests that the model isn't solely focused on learning the latent space loss. Instead, it learns to output the images as well as learn the latent space.
- d. The continuous change of beta value in the cyclical case leads to the worst performance, with the lowest PSNR score. This is possibly because the model struggles to learn effectively amidst chaotic or unstable loss function behavior.
- e. In this lab, the original model without using KL annealing exhibits the best loss and PSNR, indicating that the case doesn't require consideration of complicated noise and training problems.

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

PSNR, short for "Peak Signal-to-Noise Ratio," is a measure often used to judge how good an image or video looks after it's been compressed. It basically compares the original image or video to the compressed version and calculates how much unwanted noise or distortion is added during the compression process. The higher the PSNR value, the better the quality, because it means the original signal is much clearer compared to the added noise.

```
def Generate_PSNR(imgs1, imgs2, data_range=1.):  
    """PSNR for torch tensor"""  
    mse = nn.functional.mse_loss(imgs1, imgs2) # wrong computation for batch size > 1  
    psnr = 20 * log10(data_range) - 10 * torch.log10(mse)  
    return psnr
```

a. PSNR per frame at one of the epochs (epoch=40)



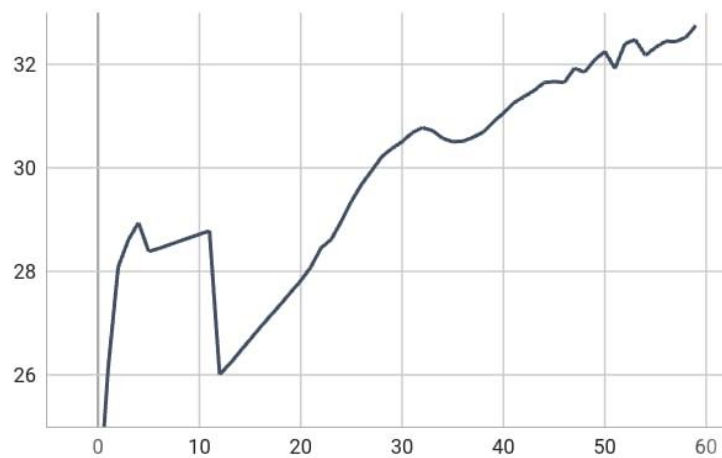
// Since the validation dataset has 650 frames; x-axis stands for number of frames

Observing the trend, it becomes apparent that the PSNR per frame decreases with each iteration, similar to a phenomenon known as error propagation. This is expected, as in each epoch, the generator takes the previously generated sequence frame as new input. Consequently, errors propagate, leading to lower PSNR values in subsequently generated outputs.

b. Average PSNR per frame over the epochs

(total epoch=60, without KL annealing, without teacher forcing)

// y-axis is average PSNR; x-axis is number of epochs



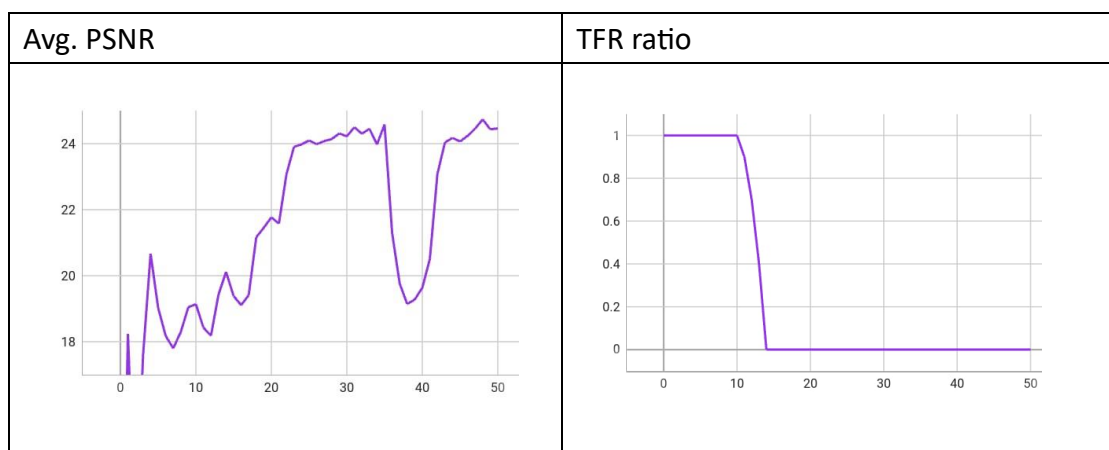
As the model undergoes training, there's a noticeable improvement in its performance as evidenced by the incremental increase in the average PSNR with each passing epoch. Moreover, the PSNR values attained on the validation dataset consistently exceed 32, mirroring the outcomes achieved in the Kaggle competition.

4. Other training strategy analysis

(1) Revision of Teacher Forcing mechanism

Traditionally, the teacher forcing ratio decreases linearly over epochs. However, I observed that altering the teacher forcing ratio during training can impact the model's learning, resulting in unstable performance (as depicted in the left graph illustrating the average PSNR over epochs below). Hence, I endeavored to reduce the interval for adjusting the ratio, yielding the updated ratio outlined in the code snippet below. With this adjustment, the model updates the ratio using the current TFR rather than 1, potentially leading to a multiplicative decrease in the ratio rather than a linear one. The modified ratio is illustrated in the graph on the right. As evident, the frequency of temporary drops in average PSNR epochs diminishes, aiding in stabilizing the model.

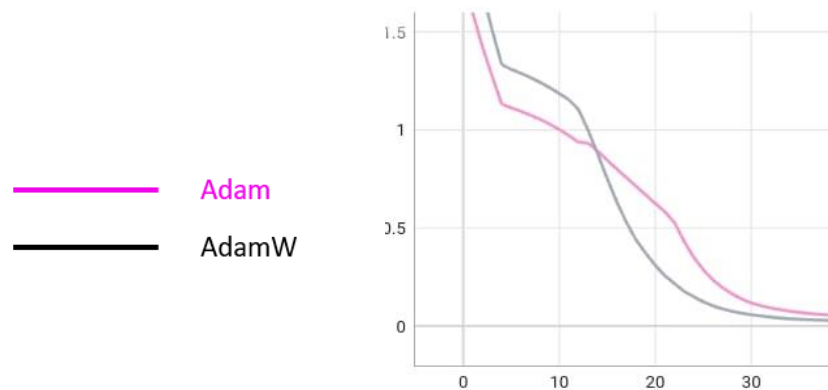
```
self.tfr = max(self.tfr-self.tfr_d_step*(self.current_epoch - self.tfr_sde), 0.0)
```



(2) Using AdamW but not Adam

Adam and AdamW are variants of the Adam optimization algorithm commonly employed in neural network training. Their primary difference lies in how they handle weight decay regularization. In standard Adam, weight decay affects both the gradient and momentum terms during parameter updates, whereas in AdamW, weight decay is applied directly to the parameters after the update step, separating it from the gradient and momentum terms. This separation in AdamW prevents overcorrection of parameters during optimization, fostering more stable training and potentially enhancing generalization performance.

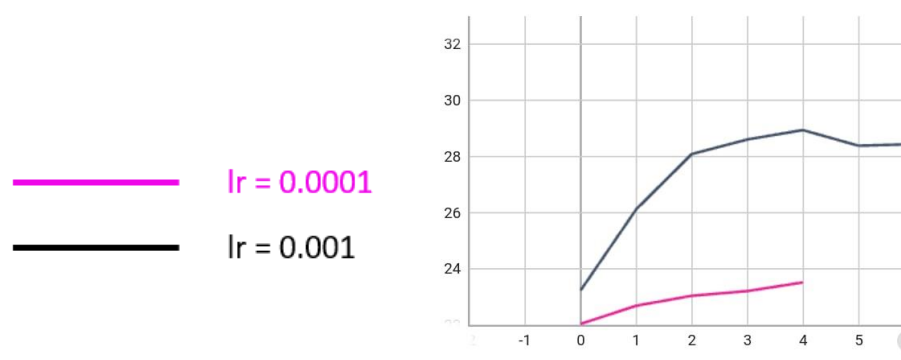
In this scenario, AdamW is superior to Adam due to its advantages in stabilizing training and improving generalization performance, particularly in scenarios where weight decay regularization is critical.



Using AdamW may have a smoother loss curve in VAE architecture.

(3) Learning rate

I discovered that when using the default learning rate of 0.001, there may be situations where the loss becomes 'nan', indicating gradient explosion in the RNN-wise architecture. Consequently, I reduced the learning rate to one-tenth of the default value. This adjustment may help prevent gradient explosion, but it also makes it more challenging for the model to converge, as the smaller learning rate may struggle to fine-tune the overall model. It's a trade-off: while this adjustment may hinder convergence, I opted to retain it to remain competitive on the Kaggle leaderboard.



The graph is about each epoch's PSNR, indicating that low learning rate may hard to converge.

5. Citation

[0] TA's spec and DL slides

[1] C. Chan, et al., "Everybody Dance Now," ICCV, 2019

[2] E. Denton, et al., "Stochastic Video Generation with a Learned Prior," ICML, 2018

[3] H. Fu, et al., "Cyclical Annealing Schedule: A Simple Approach to Mitigating KL Vanishing," NAACL 2019

[4] ChatGPT: <https://chat.openai.com>

[5] VAE video predictor github: <https://github.com/mattiasxu/Video-VQVAE>

[6] VAE deviation: <https://blog.csdn.net/a312863063/article/details/87953517>

6. The deviation of conditional VAE formula

Deviation of VAE by EM.

* $P(x) = \int_z P(z) P(x|z) dz$, where z is latent var; x is the output of decoder

⇒ find $\text{Max } P(x) \equiv \text{Max } \log P(x)$ ↓ introduce arbitrary distribution q

$$\Rightarrow \log P(x) = \int_z q(z|x) \log P(x) dz$$

$$= \int_z q(z|x) \log \left(\frac{P(z,x)}{P(z|x)} \right) dz$$

$$= \int_z q(z|x) \log \left(\frac{P(z,x)}{q(z|x)} \cdot \frac{q(z|x)}{P(z|x)} \right) dz$$

$$= \int_z q(z|x) \log \left(\frac{P(z,x)}{q(z|x)} \right) dz + \int_z q(z|x) \log \left(\frac{q(z|x)}{P(z|x)} \right) dz.$$

$$= \int_z q(z|x) \log \left(\frac{P(z,x)}{q(z|x)} \right) dz + KL(q(z|x) || P(z|x))$$

$$\star \overset{KL \geq 0}{\Rightarrow} \log P(x) \geq \int_z q(z|x) \log \left(\frac{P(z,x)}{q(z|x)} \right) dz$$

$$\Rightarrow \text{define lower bound } L_b = L(x, c, q, \theta) = \int_z q(z|x) \log \left(\frac{P(z,x)}{q(z|x)} \right) dz$$

$$\Rightarrow L = \int_z q(z|x) \log \left(\frac{P(z,x)}{q(z|x)} \right) dz$$

$$= \int_z q(z|x) \log \left(\frac{P(x|z) P(z)}{q(z|x)} \right) dz$$

$$= \int_z q(z|x) \log \left(\frac{P(z)}{q(z|x)} \right) dz + \int_z q(z|x) \log P(x|z) dz$$

$$= -KL(q(z|x) || P(z)) + \int_z q(z|x) \log P(x|z) dz.$$

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