Conditional Variational AutoEncoder for video prediction

0816095 Hsuan Wang

April 23, 2023

1 Introduction

In this lab, we are asked to implement a conditional VAE for video prediction. VAE has been applied to many problems. In our case, It should predict the next 10 frames based on the past two frames. Our goal is to be able to predict frames that has a PSNR score of more then 25, while experimenting how different implementation techniques affects the result.

2 Derivation of CVAE

Intuitively, we would want to find the parameter θ that maximize the marginal distribution $p(x;\theta)$ defined in Eq.(1).

$$p(x;\theta) = \int p(x|z;\theta)p(z)\,dz \tag{1}$$

This however becomes difficult as the integration over z is intractable when $p(x;\theta)$ is modeled by a neural network. Because we know that Maximum Likelihood = Minimize KL Divergence, we want to minimize the KL Divergence between $p(z|x;\theta)$ and $q(z|x;\theta')$.

$$\begin{aligned} KL(q(z|x;\theta')||p(z|x;\theta)) &= -\sum_z q(z|x;\theta')\log\frac{p(z|x;\theta)}{q(z|x;\theta')} \\ &= -\sum_z q(z|x;\theta')\left[\log\frac{p(x,z;\theta)}{q(z|x;\theta')} - \log p(x;\theta)\right] \\ &= -\sum_z q(z|x;\theta')\log\frac{p(x,z;\theta)}{q(z|x;\theta')} + \log p(x;\theta) \end{aligned}$$

Rearranging, we have the log likelihood equation:

$$\begin{aligned} \log p(x;\theta) &= KL(q(z|x;\theta')||p(z|x;\theta)) + \sum_z q(z|x;\theta')\log\frac{p(x,z;\theta)}{q(z|x;\theta')} \\ &= KL(q(z|x;\theta')||p(z|x;\theta)) + \mathcal{L}(x,q,\theta) \end{aligned}$$

Rearranging again, we have the equation:

$$\mathcal{L}(x,q,\theta) = \log p(x;\theta) - KL(q(z|x;\theta')||p(z|x;\theta)) \tag{2}$$

Where $KL \geq 0$, it follows that $\log p(x;\theta) \geq \mathcal{L}$. Thus $\mathcal{L}$ is called the Evidence Lower Bound, or Variational Lower Bound. By maximizing $\mathcal{L}$ instead of $\log p(x;\theta)$, we circumvent the intractable integration in Eq.(1). Next, we write variational lower bound in expectation form and have:

$$\begin{aligned}
\mathcal{L}(x,q,\theta) &= \sum_z q(z|x;\theta') \log \frac{p(x,z;\theta)}{q(z|x;\theta')} \\
&= \mathop{\mathbb{E}}_{z\sim q(z|x;\theta')} [\log p(x,z;\theta) - \log q(z|x;\theta')] \\
&= \mathop{\mathbb{E}}_{z\sim q(z|x;\theta')} [\log p(x|z;\theta) + \log p(z) - \log q(z|x;\theta')] \\
&= \mathop{\mathbb{E}}_{z\sim q(z|x;\theta')} [\log p(x|z;\theta)] + \mathop{\mathbb{E}}_{z\sim q(z|x;\theta')} \left[\log \frac{p(z)}{q(z|x;\theta')}\right] \\
&= \mathop{\mathbb{E}}_{z\sim q(z|x;\theta')} [\log p(x|z;\theta)] - KL(q(z|x;\theta')||p(z))
\end{aligned}$$

For CVAE, just condition everything on c and we have:

$$\mathcal{L}(x,q,\theta) = \mathop{\mathbb{E}}_{z\sim q(z|x,c;\theta')} [\log p(x|z,c;\theta)] - KL(q(z|x,c;\theta')||p(z|c)) \tag{3}$$

There are several ways to specify the conditional prior $p(z|c)$:

- Learn from data using a neural network
- Use a simple fixed prior without regard to c
- Ignore the regularization term (no longer VAE)

In this Lab we used a simple fixed prior(standard Gaussian).

3 Implementation details

3.1 Encoder and Decoder

The model that I used is from the sample code, which is a VGG64, which is a deep convolutional neural network. In practice, we achieve it by stacking vgg_layers. The encoder first down sample the input(which is a three channel 3x64x64 color image) to 128x1x1(which is a 1x1 image with the channel number of predefined dimension g_dim), which will then be upsampled by the decoder back to 3x64x64.

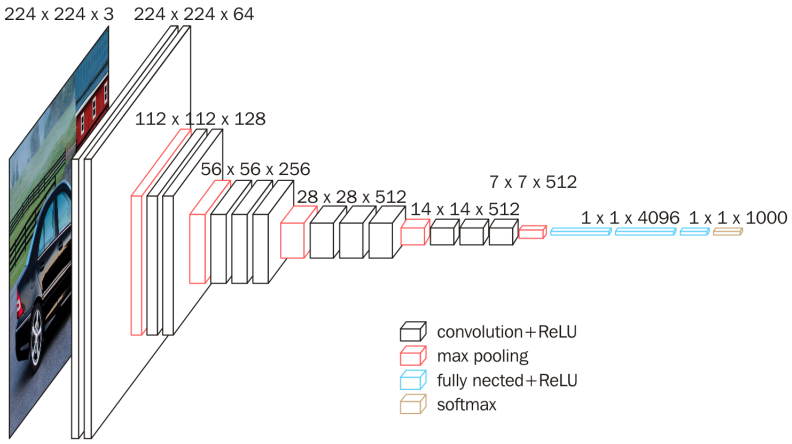

Figure 1: VGG16's architecture

3.2 Reparameterization trick

Because the act of sampling from the distribution output by the encoder is not differentiable, we introduce a technique called the reparameterization trick. Instead of sampling from the distribution itself, we can first sample from a standard normal, add the mean then multiply by variance of the original distribution, the result is the same of directly sampling from the distribution. Because we move the sampling to a standard normal which we are not going to back propagate to, we can keep the back propagation flow to the encoder.

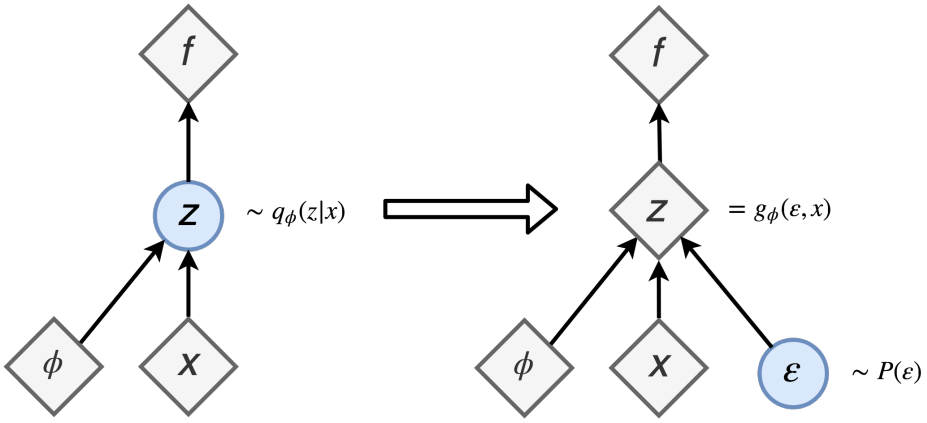

Figure 2: reparameterization trick

```
def reparameterize(self, mu, logvar):
        try:
            logvar = logvar.mul(0.5).exp_()
            eps = Variable(logvar.data.new(logvar.size()).normal_())
            return eps.mul(logvar).add_(mu)
        except:
            raise NotImplementedError
```

3.3 Dataloader

Dataloader mainly has two parts, get_seq and get_csv. get_seq loads the images in our desireable form, through the variable "ordered", we can decide whether we want to dataset to be randomly loaded or not, usually this is set to false when training(random) and set to true when testing and evaluating. In get_csv, we read the labeled action and positions from the csv file as our condition term, we will then concatenate these condition term with latent variable when training and testing.

```
    def get_seq(self):
        if self.ordered:
            self.cur_dirs = self.dirs[self.d]
            if self.d == len(self.dirs) - 1:
                self.d = 0
            else:
                self.d += 1
        else:
            self.cur_dirs = self.dirs[np.random.randint(len(self.dirs))]
        image_seq = []
        for i in range(self.frame_num):
            fname = f'{self.cur_dirs}/{i}.png'
            im = self.transform(Image.open(fname)).reshape((1, 3, 64, 64))
            image_seq.append(im)
        image_seq = torch.Tensor(np.concatenate(image_seq, axis=0))
        return image_seq
```

```
    def get_csv(self):
        cond_seq = []
        actions =
        [row for row in csv.reader(open(os.path.join(f'{self.cur_dirs}/actions.csv'), newline=''))]
        positions =
        [row for row in csv.reader(open(os.path.join(f'{self.cur_dirs}/endeffector_positions.csv'), newline=''))]
        for i in range(self.frame_num):
            concat = actions[i]
            concat.extend(positions[i])
            cond_seq.append(concat)
        cond_seq = torch.Tensor(np.array(cond_seq, dtype=float))
        return cond_seq
```

3.4 Teacher Forcing

Teacher forcing is a common training technique in sequential prediction, where instead of taking your last output as the next input, you take the ground truth as input. The benefit of doing this is that the model isn't as susceptible to false prediction in the early stages, thus achieving more stable and faster convergence. The draw back is also clear, because ground truth will not be available in testing phase, over relying on the ground truth when training will result in over-fitting, performance will drop significantly when testing data is somewhat different from the training data.

3.5 KL Cost Annealing

The KL divergence term $KL(q(z|x)||p(z))$ reaches its global minimum of 0 when $q(z|x) = p(z)$, meaning that posterior has deteriorate to the form of prior(Standard Normal). Next, when decoder $p(x|z)$ is powerful enough, it no longer need the information from z, this is called the problem of KL vanishing. To solve this problem we can dynamically adjust the weight of the KL term. In this lab, I used both Monotonic and Cyclical cost annealing.

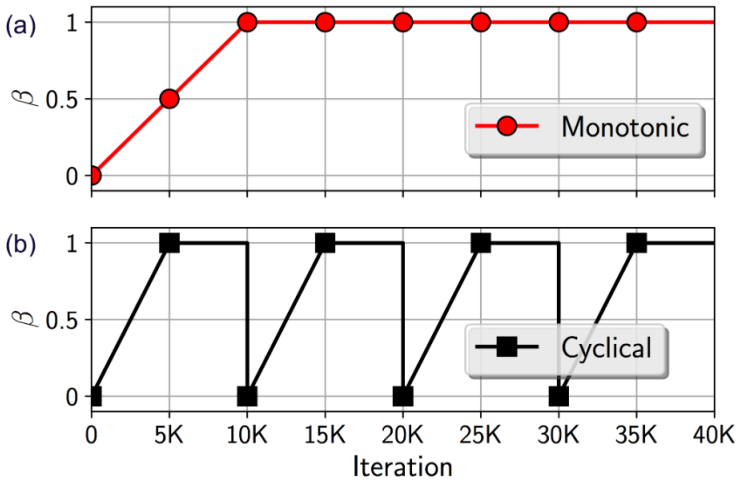

Figure 3: KL Annealing

4 Results and discussion

4.1 Show your results of video prediction

Example gif: `https://drive.google.com/file/d/1TRked3eG4XSAIr3XmqOrXv3h9ZgP3xnQ/view?usp=share_link`

```
 - Epoch: 150
 - tfr_start_decay_epoch: 15
 - tfr_decay_step: 0.01
 - tfr_lower_bound: 0.0
 - kl_anneal_cyclical: True
 - kl_anneal_cycle: 4

-> Best PSNR: 26.350490630312834
```

Figure 4: Sceenshot of best result

	tfr_start_decay_epoch = 15	tfr_start_decay_epoch = 0	w/o tf
kl_anneal_cyclical = True	25.39	25.83	25.30
kl_anneal_cyclical = False	25.33	25.38	19.48

Table 1: Comparison of performances

Figure 5: kl_anneal_cyclical = True, tfr_start_decay_epoch = 15

Figure 6: kl_anneal_cyclical = True, tfr_start_decay_epoch = 0

Figure 7: kl_anneal_cyclical = False, tfr_start_decay_epoch = 15

Figure 8: kl_anneal_cyclical = False, tfr_start_decay_epoch = 0

Figure 9: kl_anneal_cyclical = True, tfr = 0

Figure 10: kl_anneal_cyclical = False, tfr = 0

4.2 Plot the KL loss and PSNR curves during training

I tried five different set of hyperparameters, comparing the affects of Cyclical KL cost annealing and teacher forcing, below are the training process.

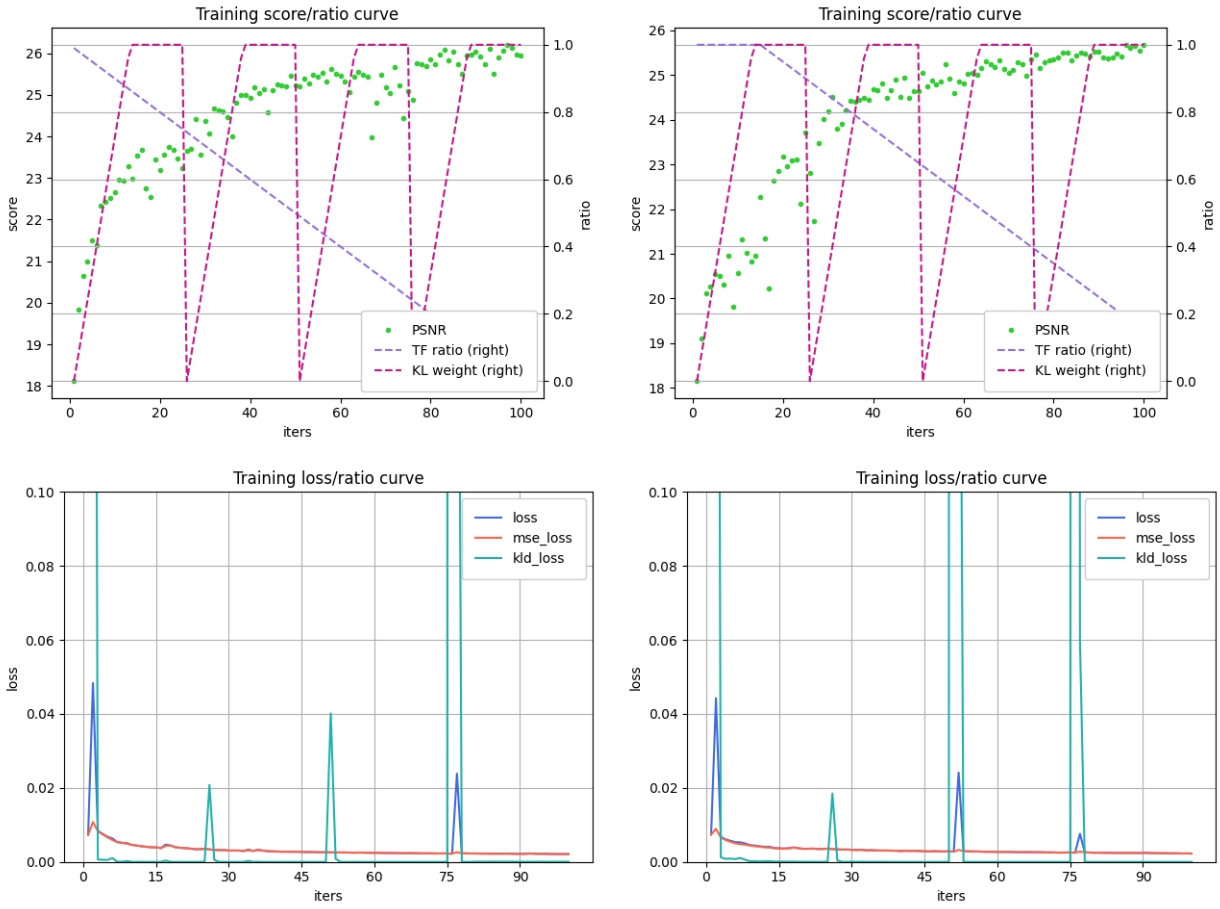

Figure 11: Cyclical kl annealing setup with different teacher forcing method

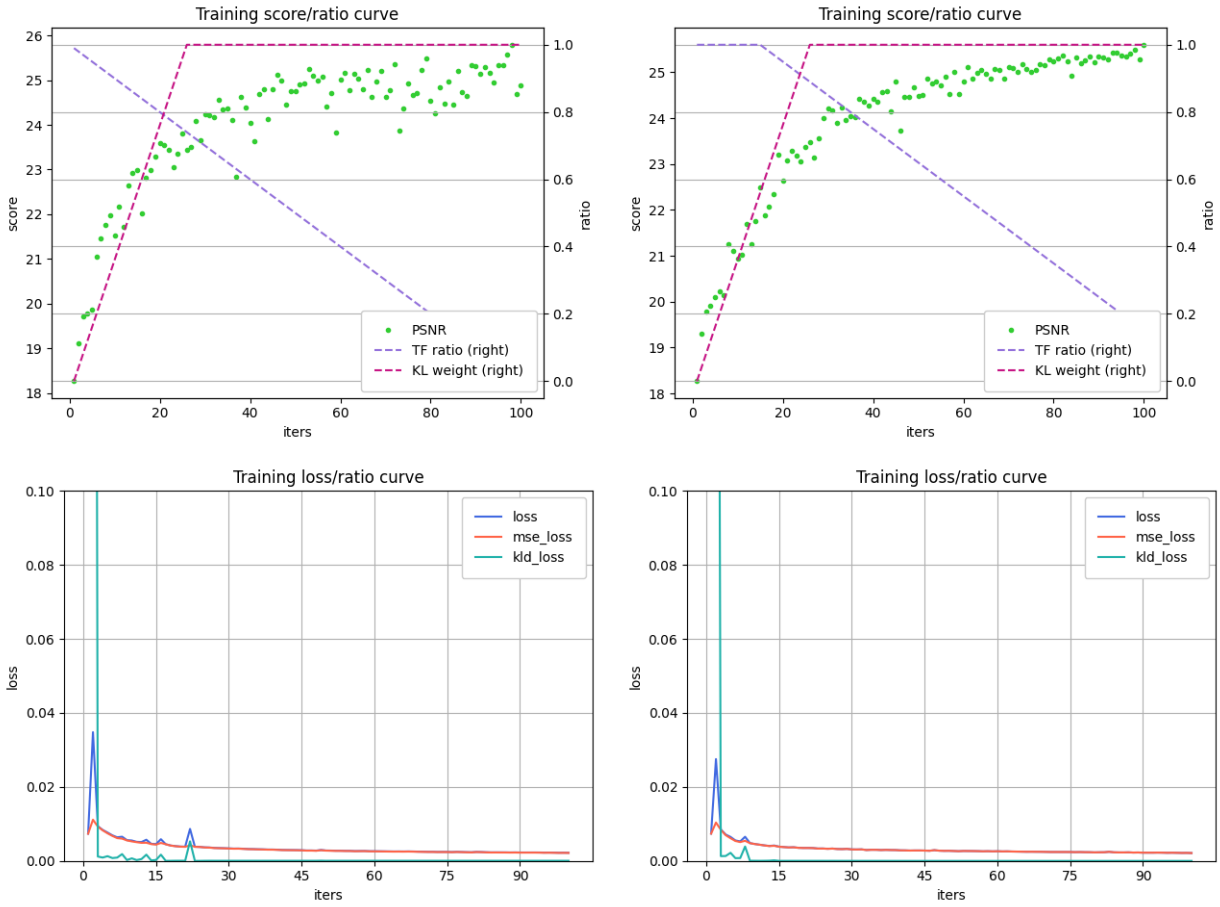

Figure 12: Monotonic kl annealing setup with different teacher forcing method

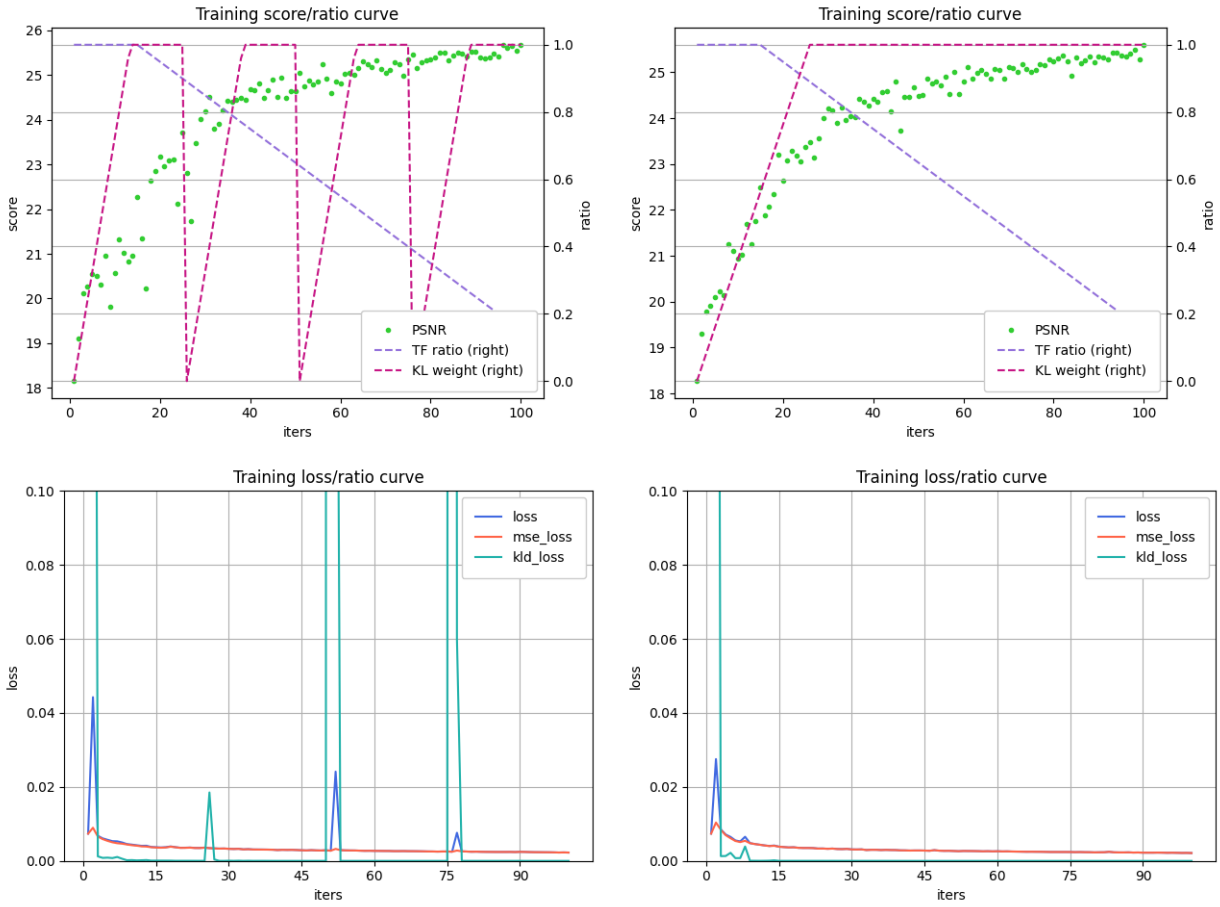

Figure 13: Teacher forcing start decaying after 15 epoch with different kl annealing method

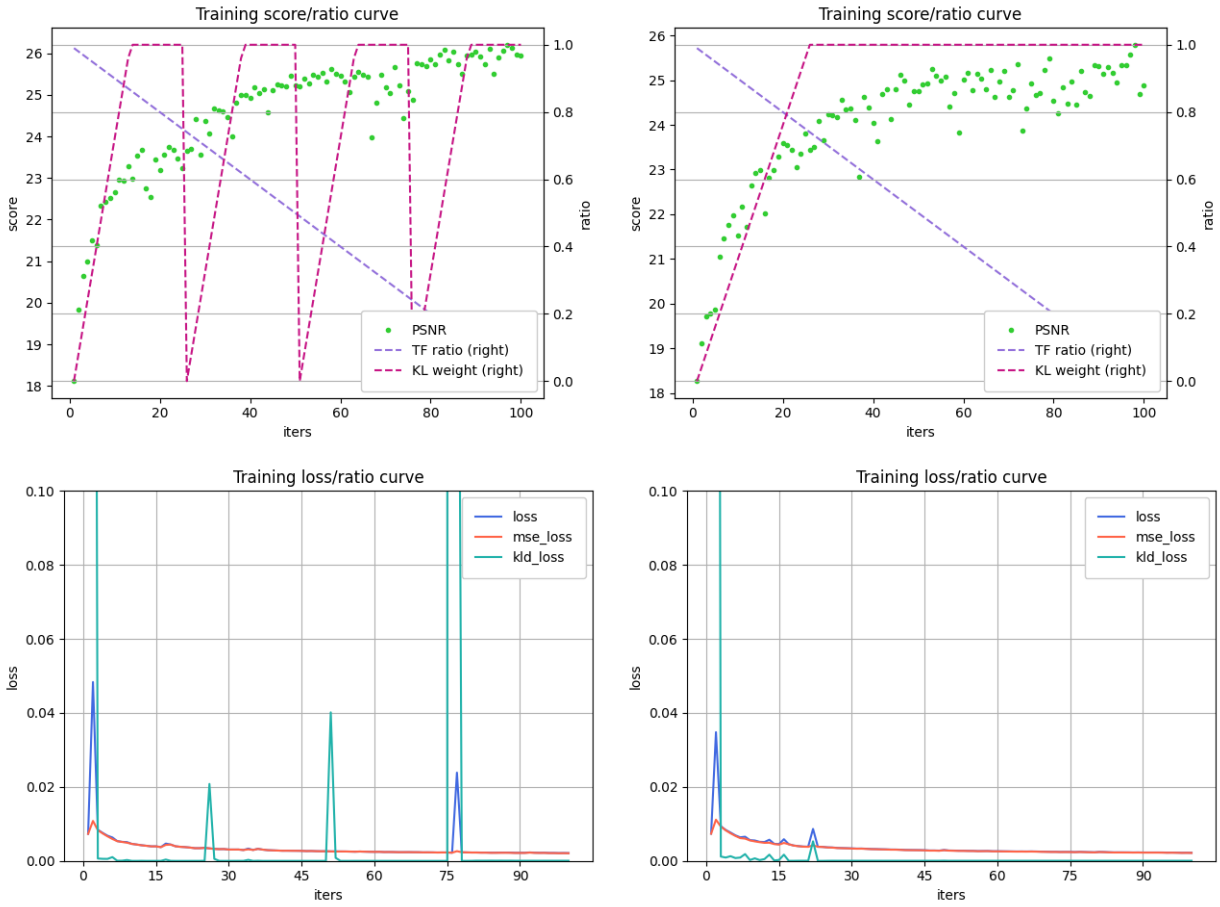

Figure 14: Teacher forcing start decaying at start with different kl annealing method

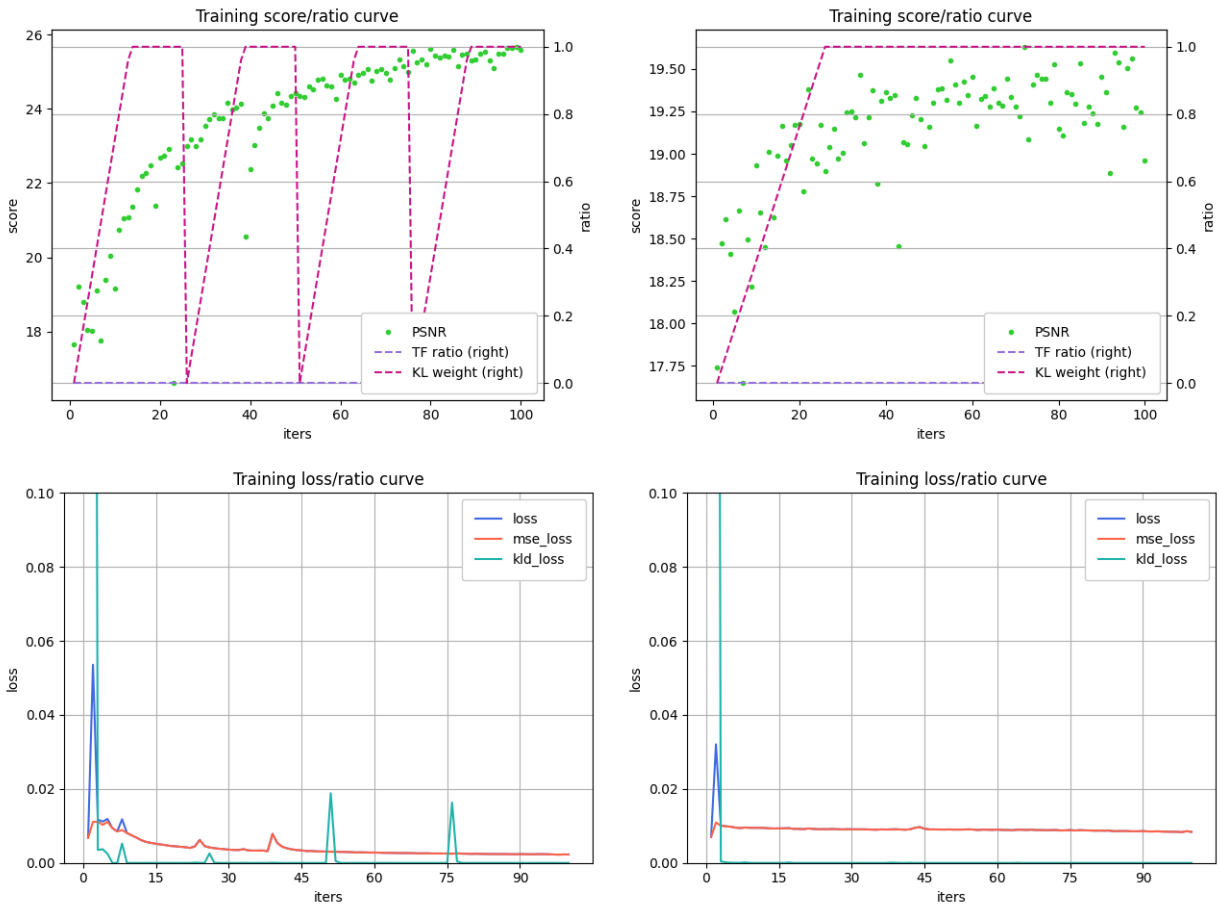

Figure 15: No teacher forcing

4.3 Discuss

From figure 10 and 11, we can see that either cyclical or monotonic kl annealing setup, start decaying teacher forcing at the very start of the training will result in a more unstable performance.

From figure 12 and 13, we can see the same situation with KL annealing, cyclical annealing generally has a more stable training process, especially when there is less teacher forcing.

Figure 14 further proves my point, when monotonic kl annealing meets 0 teacher forcing, the performance drops significantly, moreover, the training process is extremely unstable.

Although in theory Cyclical annealing and late decay teacher forcing is the most stable when training, it is not the best performing when testing. I only trained each method with 100 epoch, which is far below the model's potential capability. Because training to their full potential is also far above my graphics card and time's capability, I chose to test multiple different method with only 100 epoch. I think when trained for longer periods, Cyclical annealing with teacher forcing will out perform all.