Deep Learning Lab4

110550085 房天越

A. Introduction

In this lab, we aim to implement a model of Conditional VAE to do the task of video prediction.

Two inputs are given while training, which are the image of the previous frame and the pose image of current frame. Our task is to predict the image using the image of the current time. The given dataset is 23k video frames, and we test it 5 video sequences with 630 frames.

Our final goal is to make the prediction as similar as possible to the ground truth. In order to do so, we need to use different Teacher Forcing and KL Annealing to help us train our model. Finally, the score would be evaluated using PSNR.

B. Implementation Details

1. How do you write your training/testing protocol

   In the training part, for every batch, we first generate the predicted frame with the combination of the current frame and pose, then we calculate the MSE Loss with this predicted frame and the ground truth frame. Then for the following frames, we use the predicted frame to replace the ground truth frame to make the combination the new input, and we calculate the loss until the 15^{th} frame.

   If teacher forcing is enabled, we replace the i^{th} predicted frame as its ground truth frame. And for KL annealing, we first get the beta value, and finally when we calculate the loss function, we multiply it by the kl divergence.

   Since we don't have to generate the predicted frame of the 0^{th} frame, we use the combination of the following 15 frames and their pose to calculate kl divergence.

```python
def training_one_step(self, img, label, adapt_TeacherForcing):
    # TODO
    batch_size = img.shape[0]
    losses = 0
    beta = self.kl_annealing.get_beta()
    # print("beta: ", beta)

    for i in range(batch_size):
        im = img[i]
        la = label[i]
        x_h = im[0].unsqueeze(0)

        mse = 0
        kl_div = 0

        for j in range(1, im.size(0)):
            if adapt_TeacherForcing:
                last_im = im[j-1].unsqueeze(0)
            else:
                last_im = x_h

            im_encoder = self.frame_transformation(im[j].unsqueeze(0))
            label_encoder = self.label_transformation(la[j].unsqueeze(0))

            z, mu, logvar = self.Gaussian_Predictor(im_encoder, label_encoder)

            last_im_encoder = self.frame_transformation(last_im).detach()
            decoder_out = self.Decoder_Fusion(last_im_encoder, label_encoder, z)
            x_h = self.Generator(decoder_out)

            mse += self.mse_criterion(x_h, im[j].unsqueeze(0))
            kl_div += kl_criterion(mu, logvar, batch_size)

        loss = mse + beta * kl_div
        self.optim.zero_grad()
        loss.backward()
        self.optimizer_step()
        losses += loss

    losses /= batch_size
    return losses
```

For the testing part, the main difference is that we use the frame we generated, and we use the randn_like latent variable as the input to increase the variety of the generation.

```python
for i in range(1, 630):
    im = self.frame_transformation(decoded_frame_list[i-1].to(self.args.device))
    la = self.label_transformation(label[i].to(self.args.device))
    z, _ , _ = self.Gaussian_Predictor(im, la)

    z_lk = torch.randn_like(z)
    decoder = self.Generator(self.Decoder_Fusion(im, la, z_lk))
    decoded_frame_list.append(decoder.cpu())
    label_list.append(label[i].cpu())
```

2. How do you implement reparameterization tricks

   For this part, assume that the outputs of the model are mu and log var, we first get the variance by multiplying the logvar by 0.5 and take the exponential. Then, we get a sample from N(0, 1) with randn_like, then multiply the sample by std and plus mu, we get the N(mu, var).

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

3. How do you set your teacher forcing strategy

   For this part, we update the teacher forcing ratio by given parameters, for every tfr_sde epochs, we decrease the tfr by tfr_d_step. We need to make the dfr 0 if it decreases to less than 0.

```python
def teacher_forcing_ratio_update(self):
    # TODO

    if self.current_epoch >= self.tfr_sde:
        if self.current_epoch % self.tfr_sde == 0:
            self.tfr -= self.tfr_d_step
            if self.tfr < 0:
                self.tfr = 0
```

4. How do you set your kl annealing ratio

   In the kl annealing part, three kinds of annealing type are use, which are Cyclical, Monotonic and None (without annealing).

   For the function to generate beta by the number of iteration and cycles (frange_cycle_linear), we use the concept from the original paper.

```python
def __init__(self, args, current_epoch=0):
    # TODO
    self.iter = current_epoch + 1
    self.kl_anneal_type = args.kl_anneal_type

    if self.kl_anneal_type == 'Cyclical':
        self.L = self.frange_cycle_linear(args.num_epoch, start=0.0, stop=1.0, n_cycle=args.kl_anneal_cycle, ratio=args.kl_anneal_ratio)
    elif self.kl_anneal_type == 'Monotonic':
        self.L = self.frange_cycle_linear(args.num_epoch, start=0.0, stop=1.0, n_cycle=1, ratio=args.kl_anneal_ratio)
    elif self.kl_anneal_type == 'None':
        self.L = np.ones(args.num_epoch+1)
```

```python
def frange_cycle_linear(self, n_iter, start=0.0, stop=1.0,  n_cycle=1, ratio=1):
    # TODO
    # Adapted from the Repo of the Paper of Cyclical Annealing
    # https://github.com/haofuml/cyclical_annealing
    L = np.ones(n_iter+1)*stop
    period = n_iter // n_cycle
    step = (stop - start) / (period * ratio)
    for cycle in range(n_cycle):
        v = start
        i = 0
        while v <= stop and int(i+(cycle*period)) < n_iter:
            L[i+(cycle*period)] = v
            v += step
            i += 1

    return L
```

C. Analysis and Discussion

1. Plot Teacher Forcing Ratio

The following three experiments are cyclical, with:

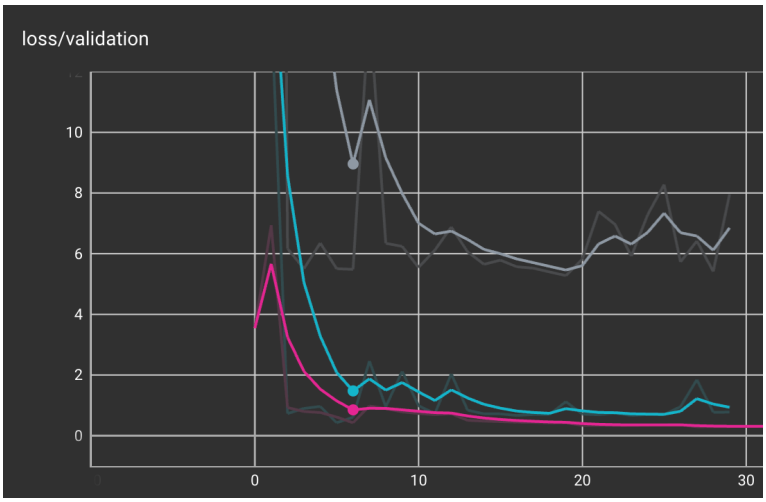
tfr = 1.0, decay = 0.1 (Gray),

tfr = 0.1, decay = 0.0 (Blue),

tfr = 0.0, decay = 0.1 (Pink, No Teacher Forcing).

By observing the validation loss, we can see that the larger tfr is, the worse performance the model has.

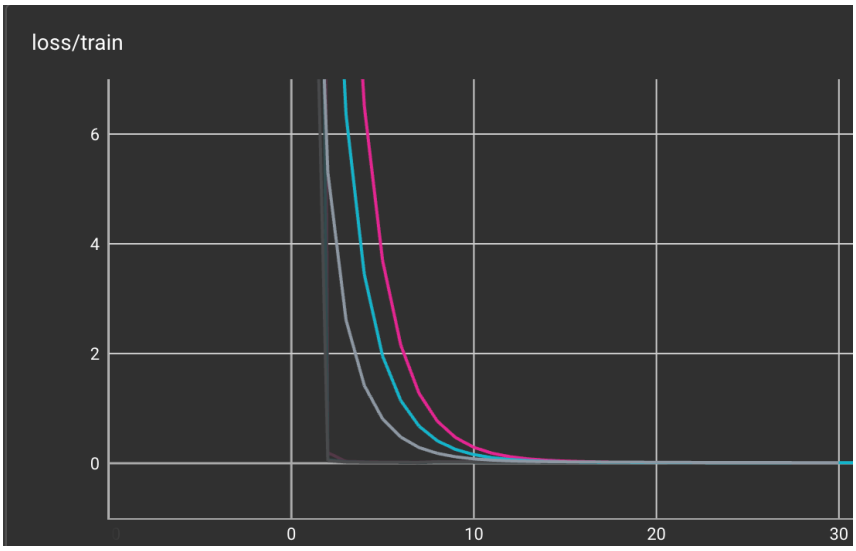
This is possibly because for every epoch, it either use teacher forcing or not for the whole epoch, and vibration generates between them. Also, it might be because that for the one with teacher forcing, the model might fall into the local minimum and converges too early, thus yields a worse performance.



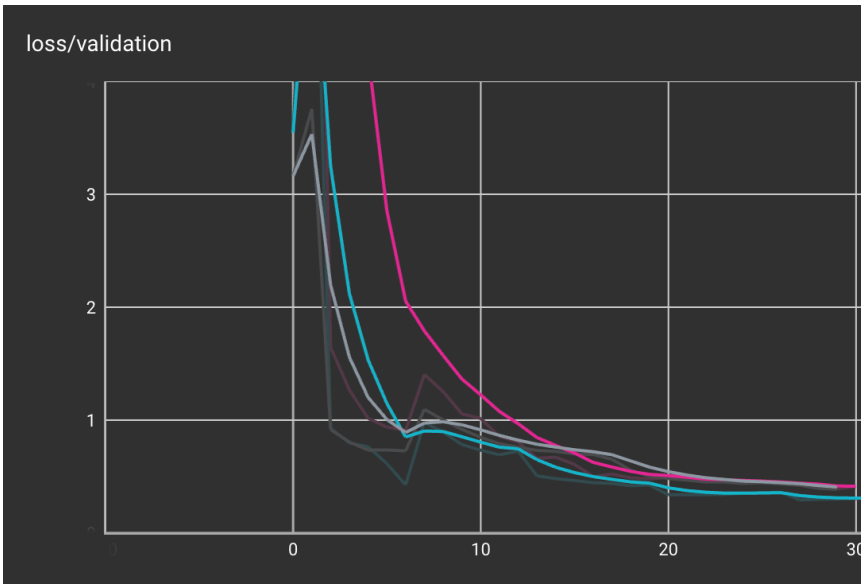
In the following graphs, light blue is Cyclical, Gray is Monotonic, and Pink is None, the tfr is set to 0, 30 epochs are used.

2. Plot the loss curve while training with different settings

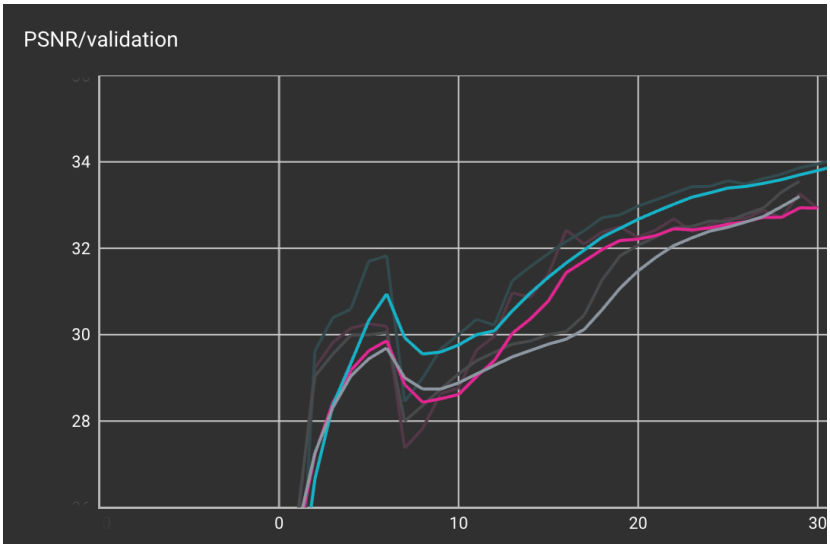
   For the training loss, we can see that all three methods converge to near 0 with the speed Monotonic>Cyclical>None, but all converge well.



However, for the validation part, we can observe that Cyclical has the best performance among the three methods, and the methods with kl annealing enabled show faster convergences on validation set.



3. Plot the PSNR-per-frame diagram in the validation dataset
   From this, we can see that the three methods have a performance that are almost the same, the Cyclical one is slightly better than None or Monotonic.



4. Other training strategy analysis

   In the original Generator code, the results are generated by the Conv2d, the result is not limited between [0, 1]. Here we add a sigmoid layer to make the output be limited between 0 and 1. Thus, we can also make the training process more stable.

```
class Generator(nn.Sequential):
    def __init__(self, input_nc, output_nc):
        super(Generator, self).__init__(
            DepthConvBlock(input_nc, input_nc),
            ResidualBlock(input_nc, input_nc//2),
            DepthConvBlock(input_nc//2, input_nc//2),
            ResidualBlock(input_nc//2, input_nc//4),
            DepthConvBlock(input_nc//4, input_nc//4),
            ResidualBlock(input_nc//4, input_nc//8),
            DepthConvBlock(input_nc//8, input_nc//8),
            nn.Conv2d(input_nc//8, 3, 1),

            nn.Sigmoid() #limit the output to [0,1]
        )
```