Lab5 Conditional VAE For Video Prediction

Deep Learning and Practice

資科工碩1 游騰德 310551054

Introduction

This lab is to implement a conditional VAE for video prediction. The model is to do prediction based on past frames. The networks to be implemented will be trained and tested on bair robot pushing small dataset. This dataset contains 44000 sequences of robot pushing motions, and each sequence include 30 frames. The main goal is to use 2 past frames to predict the next 10 frames. Additionally, the dataloader, reparameterization trick, KL annealing, and teacher forcing are required to be utilized in this lab. Finally, KL loss and PSNR curves will be calculated and shown to evaluate models performance. Please refer to the following of the report for further implementing details.

Derivation of CVAE

The chain rule of probability suggests

$$\log p(x;\theta) = \log p(x,z;\theta) - \log p(z|x;\theta)$$

Therefore, in CVAE,

$$\log P(x|c;\theta) = \log P(x,z|c;\theta) - \log P(z|x,c;\theta)$$

$$\int q(z|x,c;\theta) \log P(x|c;\theta)\, dz$$

$$= \int q(z|x,c;\theta) \log P(x,z|c;\theta)\, dz - \int q(z|x,c;\theta) \log P(z|x,c;\theta)\, dz$$

$$= \int q(z|x,c;\theta) \log P(x,z|c;\theta)\, dz - \int q(z|x,c;\theta) \log q(z|x,c;\theta)\, dz$$

$$+ \int q(z|x,c;\theta) \log q(z|x,c;\theta)\, dz - \int q(z|x,c;\theta) \log P(z|x,c;\theta)\, dz$$

$$= L(x,c,q,\theta) - KL(q(z|x,c;\theta) \| P(z|x,c;\theta)$$

$$L(x,c,q,\theta) = \log P(x|c;\theta) - KL(q(z|x,c;\theta) \| P(z|x,c;\theta)$$

$$= \int q(z|c) \log P(x|z,c;\phi)\, dz + \int q(z|c) \log q(z|c;\phi)\, dz$$

$$- KL(q(z|x,c;\theta) \| P(z|x,c;\theta)$$

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

Implementation details

Encoder and Decoder

I use the VGG models from the given sample code to implement the decoder and encoder. Both of them contains several vgg layers.

```
vgg_encoder
  (c1): Sequential
    (0): vgg_layer
    (1): vgg_layer
  (c2): Sequential
    (0): vgg_layer
    (1): vgg_layer
  (c3): Sequential
    (0): vgg_layer
    (1): vgg_layer
    (2): vgg_layer
  (c4): Sequential
    (0): vgg_layer
    (1): vgg_layer
    (2): vgg_layer
  (c5): Sequential
    (0): Conv2d(512, 128, kernel_size=(4, 4), stride=(1, 1))
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): Tanh()
  (mp): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
```

```
vgg_decoder
  (upc1): Sequential
    (0): ConvTranspose2d(128, 512, kernel_size=(4, 4), stride=(1, 1))
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): LeakyReLU(negative_slope=0.2, inplace=True)
  (upc2): Sequential
    (0): vgg_layer
    (1): vgg_layer
    (2): vgg_layer
  (upc3): Sequential
    (0): vgg_layer
    (1): vgg_layer
    (2): vgg_layer
  (upc4): Sequential
    (0): vgg_layer
    (1): vgg_layer
    (0): vgg_layer
    (1): ConvTranspose2d(64, 3, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (2): Sigmoid()
  (up): UpsamplingNearest2d(scale_factor=2.0, mode=nearest)
```

```
vgg_layer
  (main): Sequential
    (0): Conv2d
    (1): BatchNorm2d
    (2): LeakyReLU
```

Reparameterization trick

A normal distribution is multiplied on variance. After that, the value is shifted by mean. Random sampling is now treated as a noise term, and it become differentiable.

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

Dataloader

The data loader reads all of the sequences from the given root directory. Each time the __getitem__() function is called, it will return the corresponding sequence and its conditions. The sequence contains n_past + n_future images, and the conditions contain n_past + n_future rows of actions and positions. In this lab, n_past + n_future is 12.

```python
class bair_robot_pushing_dataset(Dataset):
    def __init__(self, args, mode='train', transform=default_transform, root='../data'
        assert mode == 'train' or mode == 'test' or mode == 'validate'
        self.path = f'{root}/{mode}'
        self.trajs = []
        self.transform = transform
        self.seed_is_set = False
        self.n_frames = args.n_past + args.n_future

        trajs_dirs = os.listdir(self.path)
        for trajs_dir in trajs_dirs:
            if '.tfrecords' in trajs_dir:
                traj_dirs = os.listdir(f'{self.path}/{trajs_dir}')
                for traj_dir in traj_dirs:
                    self.trajs.append((trajs_dir, int(traj_dir)))

        if args.seed is not None:
            self.set_seed(args.seed)
```

```python
    def __len__(self):
        return len(self.trajs)
        # raise NotImplementedError

    def get_seq(self, index):
        seq = []
        trajs_dir, traj_dir = self.trajs[index]
        for i in range(self.n_frames):
            img = Image.open(f'{self.path}/{trajs_dir}/{traj_dir}/{i}.png')
            img = self.transform(img)
            seq.append(img)
        seq = np.stack(seq)
        return seq
        # raise NotImplementedError

    def get_csv(self, index):
        trajs_dir, traj_dir = self.trajs[index]
        actions = np.loadtxt(f'{self.path}/{trajs_dir}/{traj_dir}/actions.csv', delimi
        positions = np.loadtxt(f'{self.path}/{trajs_dir}/{traj_dir}/endeffector_positi
        csv = np.concatenate((actions, positions), axis=1)[:self.n_frames]
        return csv
        # raise NotImplementedError

    def __getitem__(self, index):
        self.set_seed(index)
        seq = self.get_seq(index)
        cond =  self.get_csv(index)
        return seq, cond
```

Teacher forcing

Teacher forcing is to use the ground truth frame instead of predicted frame to predict the next frame. Teacher is able to make the loss to converge easier. In this lab, teacher forcing is influenced by a ratio called teacher forcing ratio (tfr). The ratio

is set between 0 and 1 to control the chance of teacher forcing to be used during training.

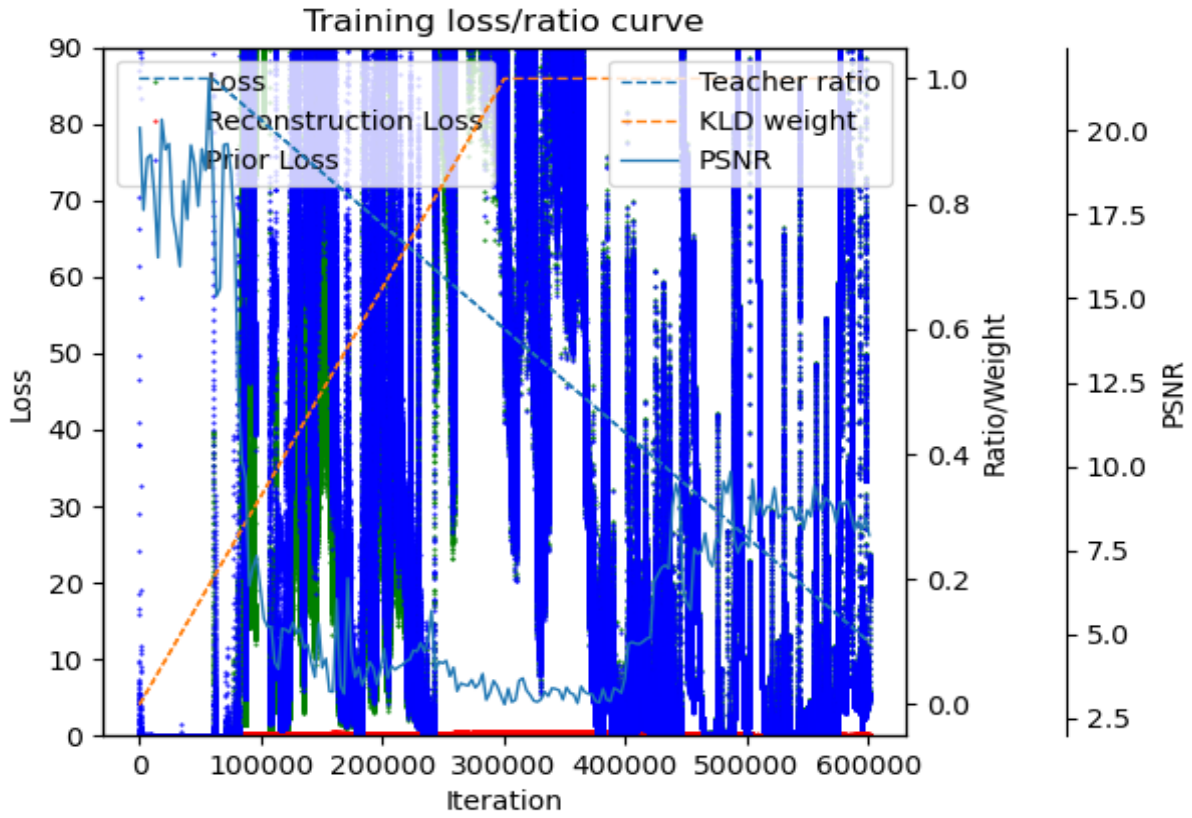
```
use_teacher_forcing = True if random.random() < args.tfr else False
```

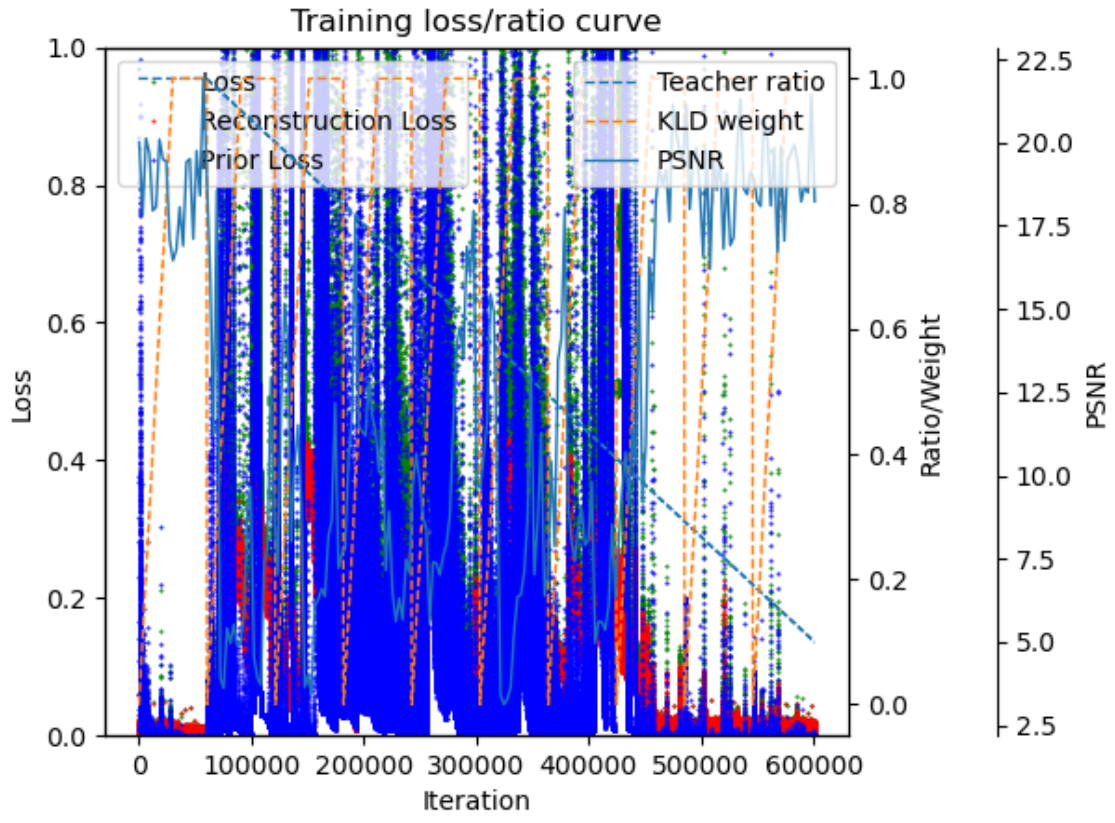
```
if use_teacher_forcing:
    next_input = x[:, i]
else:
    next_input = x_pred
```

Results

Results of video prediction

KL loss and PSNR curves





Discussion

In all trainings, I set the learning rate to 0.002. The teacher forcing ratio is set to 1.0 at the very beginning, and start to decay from epoch 100 by 0.001 per epoch. The main difference of the both training above is KL anneal method.

As we can see from the charts, when the teacher forcing ratio is set to 1.0, the loss converges very quickly. When the teacher forcing ratio starts to drop, the loss increases, and PSNR decreases.

In the training without cyclic KL annealing, PSNR decreased to about 5.0 for 300000 iterations before it increased again. However, it never go beyond 13 before the training ends, which is unsatisfactory.

On the other hand, the PSNR of the training with cyclic KL annealing increased to about 21 at the end of training, and the losses seem to converged. Meanwhile, the teacher forcing ratio is approximately 0.1. As a consequence of this result, we can say that cyclic KL annealing is beneficial to the learning of the model.

I wish to train with more different combination of hyper-parameters, such as different learning rates. However, each training last for days. Due to time and GPU resource limitations, I marked it as future explorations.