311551148 lab5 Report

1. Introduction

   VAE 是一種常見的生成模型的類別，在這次 lab，我們要實作 conditional VAE(CVAE)來做 video prediction，使用的資料即是 bair robot pushing small dataset，是一個包含 44000 sequence 的資料集，內容是機器手臂推動物體的畫面，資料及除了圖片還有提供每一個 time step 機器手臂的 action 以及 end effector position，我們的模型會以 time step = t 的圖片為輸入，搭配那個 time step 的 action, end effector position，預測 time step=t+1 的圖片，最後會使用 PSNR 來評估模型表現。

2. Derivation of CAVE

C: condition , Train model parameter θ to learn conditional distribution $P(X|c,\theta)$

$$P(X|c,\theta) = \int p(X|z,c,\theta)\, p(z|c)\, dz$$

對 z 積分 is intracable 當 $P(X|z,c,\theta)$ is model by neural network 類似 VAE, to circumvent this difficulty

$$\log P(X|c,\theta) = L(X,q,\theta,c) + KL(q(z|c) \,\|\, p(z|X,c,\theta))$$

where $L(x,q,\theta,c) = \int q(z|c) \log p(X,z|\theta,c)\, dz - \int q(z|c) \log q(z|c)\, dz$

$$KL(q(z|c) \,\|\, p(z|X,c,\theta)) = \int q(z|c) \log \frac{q(z|c)}{p(z|X,c,\theta)}\, dz$$

Reamrangement $\Rightarrow$ $L(x,q,\theta,c) = \log P(X|\theta,c) - KL(q(z|c) \,\|\, p(z|x,c,\theta))$

since the equality hold for every $q(z|c)$, we introduce a distribution $q(z|X,c,\theta')$ modeled by another neural network with parameter θ'

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

$$= E_{z\sim q(z|x,c,\theta')}[\log P(x|z,\theta,c) + \log P(z|c) - \log q(z|x,c,\theta')]$$

$$= E_{z\sim q(z|x,c,\theta')}[\log P(x|z,\theta,c) - KL(q(z|X,c,\theta') \,\|\, P(z|c))]$$

(reconstraction error) (regularization term)

$L(x,q,\theta,c)$ 為 conditional VAE 的 evidence lower bound，可以注意到所有 distribution 都 conditional on condition c

3. Implementation details

甲、Describe how you implement your model

1. Dataloader

```python
def load_train_data(args):
    print("\nLoading datasets...")

    train_data = bair_robot_pushing_dataset(args, "train")
    valid_data = bair_robot_pushing_dataset(args, "validate")

    train_loader = DataLoader(
        train_data,
        num_workers=args.num_workers,
        batch_size=args.batch_size,
        shuffle=True,
        drop_last=True,
        pin_memory=True
    )
    valid_loader = DataLoader(
        valid_data,
        num_workers=args.num_workers,
        batch_size=args.batch_size,
        shuffle=True,
        drop_last=True,
        pin_memory=True
    )

    train_iterator = iter(train_loader)
    valid_iterator = iter(valid_loader)

    return train_data, train_loader, train_iterator, \
           valid_data, valid_loader, valid_iterator
```

上面是我的 dataloader，基本上定義好 dataset 後就可以直接用 pytorch DataLoader 完成實作

ii. model

```python
def build_models(args, saved_model, device):
    print("\nBuilding models...")

    ######################
    ## Build the models ##
    ######################
    if args.model_dir != "":
        frame_predictor = saved_model["frame_predictor"]
        posterior = saved_model["posterior"]
    else:
        frame_predictor = lstm(input_size=args.g_dim + args.z_dim + args.cond_dim, output_size=args.g_dim, hidden_size=args.rnn_size,
                               n_layers=args.predictor_rnn_layers, batch_size=args.batch_size, device=device)
        posterior = gaussian_lstm(input_size=args.g_dim, output_size=args.z_dim, hidden_size=args.rnn_size,
                               n_layers=args.posterior_rnn_layers, batch_size=args.batch_size, device=device)
        frame_predictor.apply(init_weights)
        posterior.apply(init_weights)

    if args.model_dir != "":
        decoder = saved_model["decoder"]
        encoder = saved_model["encoder"]
    else:
        encoder = vgg_encoder(args.g_dim)
        decoder = vgg_decoder(args.g_dim)
        encoder.apply(init_weights)
        decoder.apply(init_weights)

    ########################
    ## Transfer to device ##
    ########################
    frame_predictor.to(device)
    posterior.to(device)
    encoder.to(device)
    decoder.to(device)

    return frame_predictor, posterior, encoder, decoder
```

model 會包含 frame_predictor, posterior, encoder, decoder，其中 framepredictor 會是 lstm 的架構，posterior 會是 gaussian lstm 的架構，除了會回傳 latent code z 之外，還會回傳 mu 以及 log variance，而 encoder 以及 decoder 都是 vgg_64 的架構，依照 argumemt dimention initial the model。

iii. reparameterization trick

```python
def reparameterize(self, mu, logvar):
    #raise NotImplementedError
    std = torch.exp(logvar / 2) ## log(variance) = log(std^2)
    eps = torch.randn_like(std) ## N(0, I) with same shape as std
    return mu + eps * std
```

reparameterization 基本上就是 z = mu + N(0,1) * std，因為模型預測的是 log(variance)所以 std = e^(logvar/2)

iv. training

```python
## Encoder all frames first
h_seq = [self.modules["encoder"](x[i]) for i in range(self.args.n_past + self.args.n_future)]

## Iterate through first 12 frames
for i in range(1, self.args.n_past + self.args.n_future):
    ## Get encoded information
    h_t, _ = h_seq[i]

    if self.args.last_frame_skip or (i < self.args.n_past):
        h_in, skip = h_seq[i - 1]
    else:
        if use_teacher_forcing:
            h_in, _    = h_seq[i - 1]
        else:
            h_in, _    = self.modules["encoder"](x_pred)

    ## Obtain latent vector z
    z_t, mu, logvar = self.modules["posterior"](h_t)

    ## Decode the image
    lstm_in = torch.cat([h_in, z_t, cond[i - 1]], dim=1)
    g_t     = self.modules["frame_predictor"](lstm_in)
    x_pred  = self.modules["decoder"]([g_t, skip])

    ## Calculate loss values
    mse = mse + mse_criterion(x[i], x_pred)
    kld = kld + kl_criterion(mu, logvar, self.args)

beta = self.kl_anneal.get_beta()
loss = mse + kld * beta
loss.backward()

self.optimizer.step()
```

上圖是 training code，我先使用 vgg encoder encode first 12 frame，接著會依序生成各個 frame，把 current frame encode representation 餵進去 gaussian lstm，得到 latent code z, mu, logvar，接著使用前一個 frame 的 encode representation，concat 上 latent code 以及 condition，餵進去 lstm 得到 prediction frame 的 encode representation，在使用 vgg decoder decode 得到 predict frame。

訓練的部分 gt_frame 以及 pred_frame 使用 mse，pred_mu, pred_logvar 使用 kl_criterion

v. testing

```
## Iterate through 12 frames
for frame_idx in range(1, args.n_past + args.n_future):
    ## Encode the image at step (t-1)
    if args.last_frame_skip or frame_idx < args.n_past:
        h_in, skip = modules["encoder"](x_in)
    else:
        h_in, _    = modules["encoder"](x_in)

    ## Obtain the latent vector z at step (t)
    if frame_idx < args.n_past:
        h_t, _    = modules["encoder"](validate_seq[frame_idx])
        _, z_t, _ = modules["posterior"](h_t) ## Take the mean
    else:
        z_t = torch.FloatTensor(args.batch_size, args.z_dim).normal_().to(device)

    ## Decode the image based on h_in & z_t
    if frame_idx < args.n_past:
        modules["frame_predictor"](torch.cat([h_in, z_t, cond[frame_idx - 1]], dim=1))
        x_in = validate_seq[frame_idx]
    else:
        g_t  = modules["frame_predictor"](torch.cat([h_in, z_t, cond[frame_idx - 1]], dim=1))
        x_in = modules["decoder"]([g_t, skip])

    pred_seq.append(x_in)
```

若是前兩個 frame，encode last frame，使用 last frame encode representation 餵進去 gaussian lstm 得到 latent code z，若不是前兩個 frame，latent code 直接 sample from N(0, 1)， 接著使用 frame predict 餵進去 encode representation, zt, condition，得到 pred frame representation，最後使用 decoder 得到圖片

vi. kl annealing

```
def update(self, epoch):
    #raise NotImplementedError

    if (epoch % self.period) <= (self.period / self.kl_anneal_ratio):
        ## Reset if cycle reached when in cyclical mode
        #if self.kl_anneal_cyclical:
        if epoch % self.period == 0:
            self.beta = 0
        else:
            self.step = (1 - 0) / (self.period / self.kl_anneal_ratio)
            self.beta = self.beta + self.step
    else:
        self.beta = 1

def get_beta(self):
    #raise NotImplementedError
    return self.beta
```

基本上就是在每個 epoch 都去 update beta，看設定是要 periodicly update 或是 monotonic

1. teacher forcing

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

```
if use_teacher_forcing:
    h_in, _     = h_seq[i - 1]
else:
    h_in, _     = self.modules["encoder"](x_pred)
```

基本上就是按照設定的機率看要不要使用 teacher forcing，若有適用就直接用前一個 Frame encode representation 當下一個 frame predict 時的 h_in，若沒有使用則是要使用上一個 predict 的 frame 的 encode representation 當 h_in

2. Describe the teacher forcing

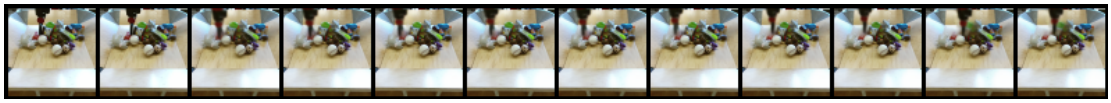
   Teacher forcing is a training strategy commonly used in auto-regressive generative models. When training models on sequential data such as time series or sentences, an intuitive way is to take the output of step t-1 as the input of step t. By adopting teacher forcing, we instead feed the ground-truth of step t-1 to the model at step t. With this approach, models often converge in less training steps.

4. Result and discussion

甲、Show your result of video prediction

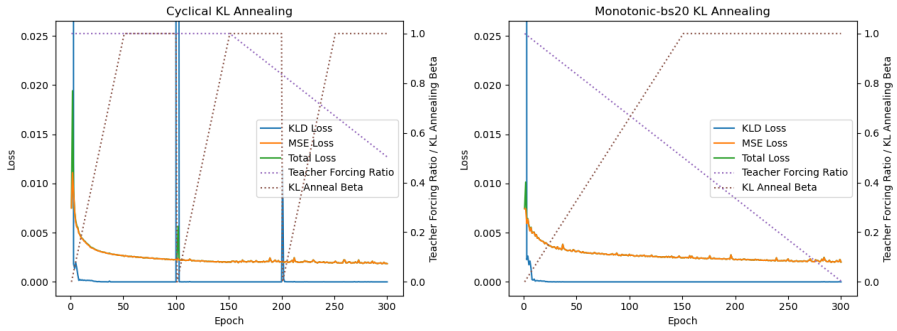
i. Output the prediction at each time step

Monotonic KL annealing

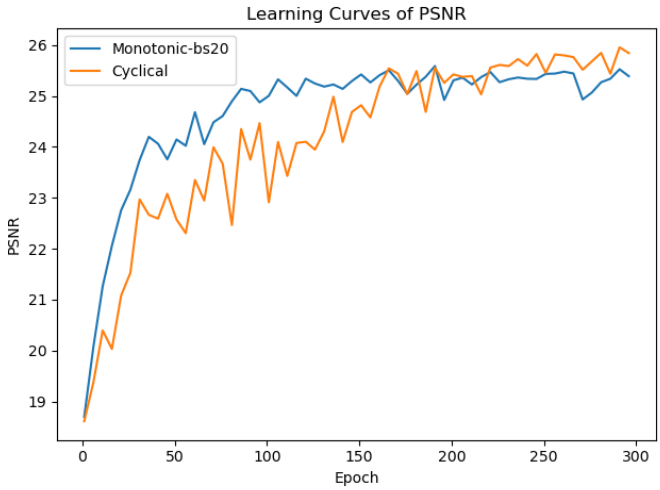


Cyclical KL annealing

乙、Plot the KL loss and PSNR curves during training



PSNR curve



PSNR 另外畫圖，因為他的值域跟前面的參數差很多

丙、Describe the results according to your setting of teacher forcing ratio, KL weight, and learning rate

KL annealing	Validation PSNR	Test PSNR
Monotonic	25.1211	25.09560
cyclical	25.6862	25.53096

上表是使用不同的 kl annealing 他們各自的 validation PSNR 以及 test PSNR，可以看到 cyclical kl annealing 有比較好的表現，另外從 KL loss curve 可以看到要是 KL weight 被設成 0 的時候，KLD 都非常的大，並且只要開始增加 KL weight，KLD 會以非常快的速度收斂到趨近於零，可以看出 KLD 對 encoder, lstm 影響非常的大，要是 KLD weight 一開始就=1，很有可能會讓模型無法 encode 完整圖片的資料到 ht，也可能會讓模型無法正確預測 mu, logvar，進而影響到後續的 perdiction。