Deep Learning Lab5 - Conditional VAE For Video Prediction Report

智能所 311581006 林子淩

1. Introduction

本次 lab 主題為實作 **V**ariational **A**uto**E**ncoder (**VAE**)。VAE 是一種被廣泛應用於 computer vision task 的生成模型，本次 lab 要求實作一個 **c**onditional **VAE** (**CVAE**)用來完成 video prediction 任務。在實驗中，首先使用 bair robot pushing 這個資料集來訓練模型，這個資料集包含約 44,000 個 robot pushing motions 的訓練例子，並針對每個 timestep 另外提供 action 和 effector position 資訊當作生成模型的生成條件。整個模型被訓練以前一幀圖片(previous frame)作為輸入，並結合 action 和 effector position 作為生成條件，去預測並生成出接下來一幀圖片(following frame)。最後，以 **P**eak **S**ignal-to-**N**oise **R**atio (**PSNR**) 衡量生成圖片與實際下一幀圖片的差別，作為生成效果的量化標準。

2. Derivation of CVAE

在訓練 CVAE 時，整體想法是訓練一個模型(其參數以θ表示)，在給定條件(condition)之下(條件以c表示)，去學習 conditional distribution $p(X|c;\theta)$。將其寫成積分形式如下：

$$p(\boldsymbol{X}|c;\boldsymbol{\theta}) = \int p(\boldsymbol{X}|\boldsymbol{Z},c;\boldsymbol{\theta})p(\boldsymbol{Z}|c)\,d\boldsymbol{Z} \tag{1}$$

在公式(1)中，將 conditional distribution $p(X|c;\theta)$寫成積分形式，Z 代表 latent code，$p(X|Z;c;\theta)$代表學習從 gaussian distribution 解碼生成 visible variable X 分布的過程。為了決定模型參數θ，必須最大化$p(X|c;\theta)$；但在 VAE 設定之下，latent Z 是由 neural network 學習得出，因此其積分式子並非 close form，無法直接估算 maximum likelihood。為了解決這個問題，可以通過 Lecture 13 Linear Factor Model 講義中，p.24 提到的 chain rule of probability，將$p(X|c;\theta)$取 log 並寫成以下形式：

$$\log p(\boldsymbol{X}|c;\boldsymbol{\theta}) = \log p(\boldsymbol{X},\boldsymbol{Z}|c;\boldsymbol{\theta}) - \log p(\boldsymbol{Z}|\boldsymbol{X},c;\boldsymbol{\theta}) \tag{2}$$

針對公式(2)，等式兩邊皆引入一個 arbitrary distribution $\mathrm{q}(Z|c)$，並對 Z 積分，可得到：

$$\begin{aligned}
\log p(\boldsymbol{X}|c;\boldsymbol{\theta}) &= \int \mathrm{q}(\boldsymbol{Z}|c)\ \log p(\boldsymbol{X}|c;\boldsymbol{\theta})\ d\boldsymbol{Z} \\
&= \int \mathrm{q}(\boldsymbol{Z}|c)\log p(\boldsymbol{X},\boldsymbol{Z}|c;\boldsymbol{\theta})\,d\boldsymbol{Z} - \int \mathrm{q}(\boldsymbol{Z}|c)\log p(\boldsymbol{Z}|\boldsymbol{X},c;\boldsymbol{\theta})\,d\boldsymbol{Z} \\
&= \left\{\int \mathrm{q}(\boldsymbol{Z}|c)\log p(\boldsymbol{X},\boldsymbol{Z}|c;\boldsymbol{\theta})\,d\boldsymbol{Z} - \int \mathrm{q}(\boldsymbol{Z}|c)\log \mathrm{q}(\boldsymbol{Z}|c)\,d\boldsymbol{Z}\right\} \\
&\quad + \left\{\int \mathrm{q}(\boldsymbol{Z}|c)\log \mathrm{q}(\boldsymbol{Z}|c)\,d\boldsymbol{Z} - \int \mathrm{q}(\boldsymbol{Z}|c)\log p(\boldsymbol{Z}|\boldsymbol{X},c;\boldsymbol{\theta})\,d\boldsymbol{Z}\right\} \\
&= \mathrm{L}(\mathbf{X},q,\boldsymbol{\theta}|c) + \mathrm{KL}(\mathrm{q}(\boldsymbol{Z}|c)\ ||\ p(\boldsymbol{Z}|\boldsymbol{X},c;\boldsymbol{\theta}))
\end{aligned} \tag{3}$$

公式(3)中，$L(\mathbf{X},q,\boldsymbol{\theta}|c)$被稱為 **e**vidence **l**ower **b**ound (**ELBO**)，$KL(q(\boldsymbol{Z}|c)\ ||\ p(\boldsymbol{Z}|\boldsymbol{X},c;\boldsymbol{\theta}))$則為$q(\boldsymbol{Z}|c)$和$p(\boldsymbol{Z}|\boldsymbol{X},c;\boldsymbol{\theta})$兩者之間的 KL divergence。ELBO 和 KL term 的公式如下所示：

$$L(\mathbf{X},q,\boldsymbol{\theta}|c) = \int q(\boldsymbol{Z}|c)\log p(\boldsymbol{X},\boldsymbol{Z}|c;\boldsymbol{\theta})\,d\boldsymbol{Z} - \int q(\boldsymbol{Z}|c)\log q(\boldsymbol{Z}|c)\,d\boldsymbol{Z}$$

$$KL(q(\boldsymbol{Z}|c)\ ||\ p(\boldsymbol{Z}|\boldsymbol{X},c;\boldsymbol{\theta})) = \int q(\boldsymbol{Z}|c)\log q(\boldsymbol{Z}|c)\,d\boldsymbol{Z} - \int q(\boldsymbol{Z}|c)\log p(\boldsymbol{Z}|\boldsymbol{X},c;\boldsymbol{\theta})\,d\boldsymbol{Z}$$

$$= \int q(\boldsymbol{Z}|c)\frac{q(\boldsymbol{Z}|c)}{p(\boldsymbol{Z}|\boldsymbol{X},c;\boldsymbol{\theta})}d\boldsymbol{Z}$$

將公式(3)重新交換順序，可得：

$$L(\mathbf{X},q,\boldsymbol{\theta}|c) = \log p(\boldsymbol{X}|c;\boldsymbol{\theta}) - KL(q(\boldsymbol{Z}|c)\,||\,p(\boldsymbol{Z}|\boldsymbol{X},c;\boldsymbol{\theta})) \tag{4}$$

由於公式(4)等式對於任何$q(\boldsymbol{Z}|c)$分布都成立，因此可以使用另一個 neural network 擔任 encoder 的角色，用來學習$q(\boldsymbol{Z}|\boldsymbol{X},c;\boldsymbol{\theta}')$分布。公式如下：

$$\begin{aligned} L(\mathbf{X},q,\boldsymbol{\theta}|c) &= \log p(\boldsymbol{X}|c;\boldsymbol{\theta}) - KL(q(\boldsymbol{Z}|\boldsymbol{X},c;\boldsymbol{\theta}')\,||\,p(\boldsymbol{Z}|\boldsymbol{X},c;\boldsymbol{\theta})) \\ &= \mathbb{E}_{\mathbf{Z}\sim q(\boldsymbol{Z}|\boldsymbol{X},c;\boldsymbol{\theta}')}[\log p(\boldsymbol{X}|\boldsymbol{Z},c;\boldsymbol{\theta}) + \log p(\boldsymbol{Z}|c) - \log q(\boldsymbol{Z}|\boldsymbol{X},c;\boldsymbol{\theta}')] \\ &= \mathbb{E}_{\mathbf{Z}\sim q(\boldsymbol{Z}|\boldsymbol{X},c;\boldsymbol{\theta}')}[\log p(\boldsymbol{X}|\boldsymbol{Z},c;\boldsymbol{\theta})] - KL(q(\boldsymbol{Z}|\boldsymbol{X},c;\boldsymbol{\theta}')\,||\,p(\boldsymbol{Z}|c)) \end{aligned} \tag{5}$$

公式(5)整理並推導出 conditional VAE 模型的 evidence lower bound，可以看出所有分布皆以 condition c 作為條件。第一項期望值的部分稱作 reconstruction term，意義為模型 decoder 對 latent Z 解碼出來的結果與原始 input X 之間的 likelihood。第二項 KL divergence 部分稱作 regularization term，意義為限制 encoder 在將 input X 轉換到 latent Z 時，其分布不可以距離理想 latent 分布$p(\boldsymbol{Z}|c)$太遠，也就是 posterior distribution 受到 prior distribution 限制。

回顧前面提到，因積分式並非 close form 因此無法直接對$p(\boldsymbol{X}|c;\boldsymbol{\theta})$計算 maximum likelihood 的問題，通過公式推導發現可以用最大化 evidence lower bound $L(\mathbf{X},q,\boldsymbol{\theta}|c)$代替。而 maximize $L(\mathbf{X},q,\boldsymbol{\theta}|c)$又可以看作是去 maximize reconstruction term (用 decoder 解碼 latent 的結果與 input X 的相似程度)，同時 minimize regularization term (encoder 轉換的 latent Z 分布 $q(\boldsymbol{Z}|\boldsymbol{X},c;\boldsymbol{\theta}')$和理想中 latent 分布$p(\boldsymbol{Z}|c)$，兩個分布之間的差距。前者被稱為 encoded distribution，後者被稱為 prior distribution)。

3. Implementation details

Describe how you implement your model (encoder, decoder, reparameterization trick, dataloader, etc.)

- **Dataloader**

Figure 1, 2 是 dataloader 實作程式碼。在 Figure 1 中`__init__`定義取得 dataset 路徑(`self.root`)、每個 sequence 要取幾個 frame 當作訓練和測試使用(`self.seq_len`)，以及其他會用到的路徑和參數設定。函數 `set_seed` 將 numpy.random 的隨機性固定下來，確保實驗可再現。Figure 2 中包含了`get_seq`, `get_csv`, `__getitem__`三個主要功能函數。`get_seq` 負責按照路徑讀取 video frames sequence，並將其轉換成形狀為 (number of frames, 3, 64, 64) 的 tensor，作為模型訓練的輸入資料。`get_csv` 讀取 video sequence 對應的 action 和 effector position 資訊當作生成條件(condition)用來輔助預測，其形狀為 (number of frames, 7) 的 tensor。而`__getitem__`則是在用 dataloader 取得 batch data 時執行，並回傳 video sequence 和對應的 condition，其形狀分別為 (batch size, number of frames, 3, 64, 64) 和 (batch size, number of frames, 7)。

```python
class bair_robot_pushing_dataset(Dataset):
    def __init__(self, args, mode='train', transform=default_transform):

        assert mode == 'train' or mode == 'test' or mode == 'validate'
        self.root = '{}/{}'.format(args.data_root, mode)
        self.seq_len = max(args.n_past + args.n_future, args.n_eval)
        self.mode = mode

        self.transform = transform
        self.dirs = []
        for dir1 in os.listdir(self.root):
            for dir2 in os.listdir(os.path.join(self.root, dir1)):
                self.dirs.append(os.path.join(self.root, dir1, dir2))

        self.seed_is_set = False
        self.cur_dir = self.dirs[0]

    def set_seed(self, seed):
        if not self.seed_is_set:
            self.seed_is_set = True
            np.random.seed(seed)

    def __len__(self):
        return len(self.dirs)
```

Figure 1 dataloader part 1

```python
    def get_seq(self, index):
        self.cur_dir = self.dirs[index]
        image_seq = []
        for i in range(self.seq_len):
            fname = '{}/{}.png'.format(self.cur_dir, i)
            img = Image.open(fname)
            image_seq.append(self.transform(img))
        image_seq = torch.stack(image_seq)
        return image_seq

    def get_csv(self, index):
        self.cur_dir = self.dirs[index]
        with open('{}/actions.csv'.format(self.cur_dir), newline='') as csvfile:
            rows = csv.reader(csvfile)
            actions = []
            for i, row in enumerate(rows):
                if i == self.seq_len:
                    break
                action = [float(value) for value in row]
                actions.append(torch.tensor(action))
            actions = torch.stack(actions)

        with open('{}/endeffector_positions.csv'.format(self.cur_dir), newline='') as csvfile:
            rows = csv.reader(csvfile)
            positions = []
            for i, row in enumerate(rows):
                if i == self.seq_len:
                    break
                position = [float(value) for value in row]
                positions.append(torch.tensor(position))
            positions = torch.stack(positions)

        # concat embedding
        condition = torch.cat((actions, positions), axis=1)
        return condition

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

Figure 2 dataloader part 2

- **Reparameterization Trick**

Reparameterization trick 目的是去模擬 sample 過程，取代計算 reconstruction term 時期望值要通過抽樣取得，避免在訓練過程造成斷點沒辦法做 backpropagation。Reparameterization 公式如下：

$$\mathbf{Z} = \mu(\mathbf{X}) + \Sigma(\mathbf{X}) * \varepsilon \tag{6}$$

公式(6)中，$\mu(\mathbf{X})$和$\Sigma(\mathbf{X})$分別代表通過 gaussian LSTM (見後面 Encoder & Decoder 說明)預測出 latent Z 的 mean 和 standard deviation，$\varepsilon \sim N(0,1)$則是從 gaussian distribution (mean=0, variance=1)隨機抽樣，經過乘加運算可以模擬出從 latent Z 抽樣出來的 sample，取代真實抽樣。

Figure 3 為 reparameterization trick 的實作程式碼，`eps` 即公式中的ε。要特別注意 LSTM 預測出來的是 log variance 而非 variance，這是因為根據定義 variance 必為正實數(positive real number)，如果以 variance 作為預測目標會限制模型預測結果的範圍。通過取 log 轉換，可以將預測結果範圍由[0,1]轉換到[-∞,log(1)]，使得模型的預測效果更加穩定也更好訓練。由於以上原因，`std` 將 log variance 轉換到 standard deviation，最後組成模擬 latent 抽樣結果 z 回傳。

```
60      def reparameterize(self, mu, logvar):
61          # sample from N(0,I)
62          eps = torch.randn_like(mu)
63          # standard deviation sigma = sqrt(variance) = sqrt(exp(log variance))
64          std = torch.exp(0.5*logvar)
65          # latent code z
66          z = mu + eps * std
67          return z
```

Figure 3 reparameterization trick

- **Encoder & Decoder**

在本次實驗中，我使用 sample code 中給定的 encoder 和 decoder 架構。模型使用 VGG64 作為 video frame 的 encoder 和 decoder，其中 decoder 還包含了 skip connection 的設計，避免 gradient vanishing 問題。另外還使用兩個 LSTM 分別學習 video sequence 的 latent representation 和 posterior distribution 的資訊。第一個 LSTM 的輸入有三個，分別為 timestep = t-1 時的 video frame 經過 VGG64 encode 的結果、timestep = t 的生成條件、timestep = t 時第二個 LSTM 計算出的 posterior distribution 抽樣結果；最後輸出為 video sequence 的 latent space representation，之後輸入 VGG64 decoder 用來預測 timestep = t 的 video frame。第二個 LSTM 為 gaussian LSTM，負責學習 latent Z 的 distribution 並使其接近設定中的 prior distribution (gaussian distribution, mean=0, variance=1)，通過 reparameterization trick 模仿抽樣過程取代抽樣計算期望值，讓整個訓練過程不會出現無法做 backpropagation 的斷點。由於程式碼過於冗長，詳細請參考 source code / models 內的檔案。

- **Training Process**

Figure 4 為訓練一個 batch sequence 的實作程式碼。首先，使用 VGG encoder 去 encoder 前 12 個 frames，得到 `h_seq`，模型再在迴圈中預測出每個 timestep 對應的 video frame。在迴圈的每一個 iteration 中，模型會從 `h_seq` 取出上一個 frame 的 encoded representation h_{t-1} (`h`)和當下 timestep frame 的 encoded representation h_t (`h_target`)，將 `h_target` 送進 LSTM 裡面取得對應的 latent vector z_t (`z_target`)。再將 `h`、`z_target` 和上一個時間點的生成條件c_{t-1} (`cond`)一起送進另一個 LSTM 當作輸入，該 LSTM 輸出的 latent representation 會再送進 VGG decoder，最後預測出針對當下 timestep 預測出來的 frame prediction 結果$\hat{x}_t$，即 `x_pred`。最後計算 reconstruction error (MSE loss)和 latent z distribution 和 prior distribution 之間的 KL divergence，相加作為 backpropagation 的 loss。詳細見 source code / trainer 內的檔案。

```
242         """encode all sequence first (avoid repeatedly encode in next loop)"""
243         h_seq = [self.modules['encoder'](x[i]) for i in range(self.args.n_past + self.args.n_future)]
244
245         """predict 1~12 frames"""
246         # x_pred = None
247         for frame_idx in range(1, self.args.n_past + self.args.n_future):
248             """h = x_(t-1), h_target = x_t"""
249
250             # h_target = input for prior lstm (posterior)
251             h_target, _ = h_seq[frame_idx]
252             # h, skip = input for lstm before decoder (frame_predictor)
253             # skip last frame or condition on n_past frame (must use teacher forcing with skip)
254             if self.args.last_frame_skip or (frame_idx < self.args.n_past):
255                 h, skip =  h_seq[frame_idx-1]
256             else:
257                 if use_teacher_forcing:
258                     h, _ = h_seq[frame_idx-1]
259                 else:
260                     h, _  = self.modules['encoder'](x_pred)
261
262             # construct latent code z of h_1
263             z_target, mu, logvar = self.modules['posterior'](h_target)
264
265             # input to lstm before decoder (frame_predictor)
266             lstm_in = torch.concat([h, z_target, cond[frame_idx-1]], dim=1)
267             lstm_out = self.modules['frame_predictor'](lstm_in)
268             x_pred = self.modules['decoder']((lstm_out, skip))
269
270             # reconstruction loss
271             mse += mse_criterion(x_pred, x[frame_idx])
272             # KL divergence
273             kld += kl_criterion(mu, logvar, self.args)
```

Figure 4 Training process

- **Testing Process**

Figure 5 為 testing/validation 的實作程式碼。在迴圈中的每一個 iteration，模型首先使用 encoder，得到上一個 time step 中，video frame 的 encoded representation h_{t-1} (h)。接著，由於測試中是根據最前面兩個 frames 去預測後面接續的十個 frames，因此如果當 iteration 還在預測前兩個 frames 時，latent z (z_target)可以直接從高斯分布 N(0,1)裡抽樣得到(使用 teacher forcing)。而在預測後十個 frames 時，則使用 encoder 和 Gaussian LSTM 產生 latent code z_t (z_target)，等同於 teacher forcing ratio 設定為 0。最後，frame predictor (另一個 LSTM)會根據輸入 h、z_target 和上一個時間點的生成條件c_{t-1} (cond)，輸出結果(lstm_out)並通過 VGG decoder 預測出針對當下 timestep 預測出來的 frame prediction 結果$\hat{x}_t$，即 x_in。

```python
    """predict 1~12 frames"""
    # initial input image x_in = x[0]
    gen_seq = []
    gen_seq.append(x[0])
    x_in = x[0]
    for frame_idx in range(1, args.n_past + args.n_future):
        # encode input image h at timestep (t-1) => (frame_idx-1)
        if args.last_frame_skip or (frame_idx < args.n_past):
            h, skip =  modules['encoder'](x_in)
        else:
            h, _  = modules['encoder'](x_in)

        # get latent code z at timestep (t) => (frame_idx)
        if (frame_idx < args.n_past):
            h_target, _ = modules['encoder'](x[frame_idx])
            # z_target, mu, logvar = modules['posterior'](h_target)
            _, z_target, _ = modules['posterior'](h_target)
        else:
            # fill tensor with normal sample (mean,std)
            z_target = torch.cuda.FloatTensor(args.batch_size, args.z_dim).normal_()

        # decode and output x at timestep (t) based on h and z
        if frame_idx < args.n_past:
            # necessary, for lstm hidden state
            modules['frame_predictor'](torch.concat([h, z_target, cond[frame_idx - 1]], dim=1))
            # update x_in for next frame prediction
            x_in = x[frame_idx]
            gen_seq.append(x_in)
        else:
            lstm_in = torch.concat([h, z_target, cond[frame_idx - 1]], dim=1)
            lstm_out = modules['frame_predictor'](lstm_in)
            x_in = modules['decoder']((lstm_out, skip))
            gen_seq.append(x_in)

    gen_seq = torch.stack(gen_seq)
    return gen_seq
```

Figure 5 Testing process (for evaluation)

Describe the teacher forcing (including main idea, benefits and drawbacks.)

Teacher forcing 是一種常用在 auto-regressive generative model 的訓練策略。在針對 time series 或 sentence 這類有關於時序的 sequential data 作為訓練資料時，直觀上會將 timestep = t-1 時的輸出作為 timestep = t 的輸入，以實現按照時序訓練模型的效果。然而這樣的方法會有無法平行化運算(必須先等前一個 timestep 結果計算完畢，才可以輸入模型計算下一個 timestep 結果)，以及如果模型一開始預測效果不好，timestep = t-1 的輸出預測結果遠遠偏離真實結果，則會讓模型一直以錯誤的結果作為輸入，難以收斂。

為了解決以上問題，teacher forcing 成為重要的訓練技巧，通過將 timestep = t-1 的真實結果(ground truth)取代預測結果(prediction)作為 timestep = t 的輸入，可以讓模型在較少的訓練 epoch 中達到收斂。與此同時，使用 teacher forcing 也可以緩解誤差傳播(error propagation)的問題，error propagation 即在沒有使用 teacher forcing 的情況之下，模型可能會重複將前一個 timestep 的錯誤預測做為下一個 timestep 的 input，導致收斂速度比較慢的情況。

不過，teacher forcing 也有缺點，例如模型過度依賴有對應的 ground truth 標記的訓練資料，因此會讓模型失去部分泛化能力(generalizability)。這代表如果整個訓練過程都將 teacher forcing ratio 設定為 1，不在某些 timestep 以 prediction 作為輸入讓模型學習不靠 ground truth 做預測的話，模型可能會在預測沒有看過的資料(validation/testing data)上預測能力不佳。

4. Results and discussion

Show your results of video prediction

Table 1 顯示模型針對每個 timestep 的預測結果，gif 檔則在 source code / result 中附上。表格上方為採用 cyclical KL annealing 模式訓練模型預測出來的結果，下方為採用 monotonic KL annealing 模式訓練模型預測出來的結果。

Cyclical KL Annealing

Monotonic KL Annealing


Table 1　模型針對每個 timestep 的預測結果

Plot the KL loss and PSNR curves during training

Table 2 顯示 loss、teacher forcing ratio、KL anneal β 的 learning curve。Figure 6 則是 average PSNR 的 learning curve，因為尺度不同因此單獨畫在一張圖。

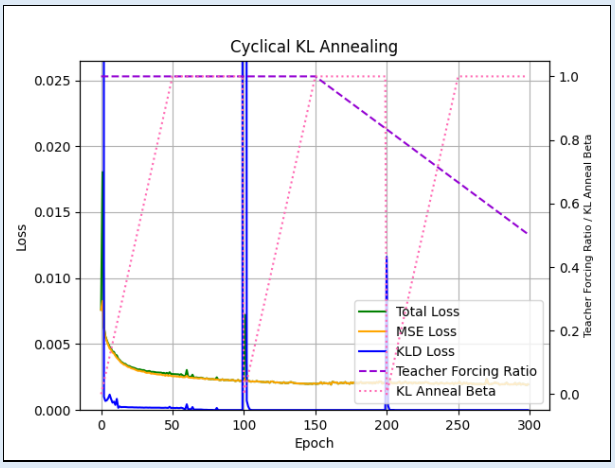
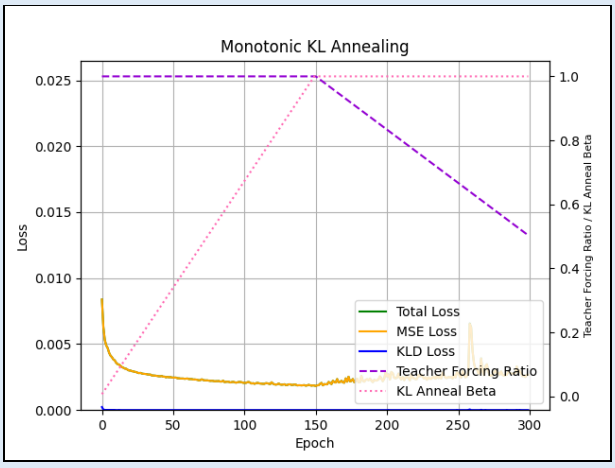
Cyclical KL Annealing	Monotonic KL Annealing
	

Table 2 loss and ratio curve

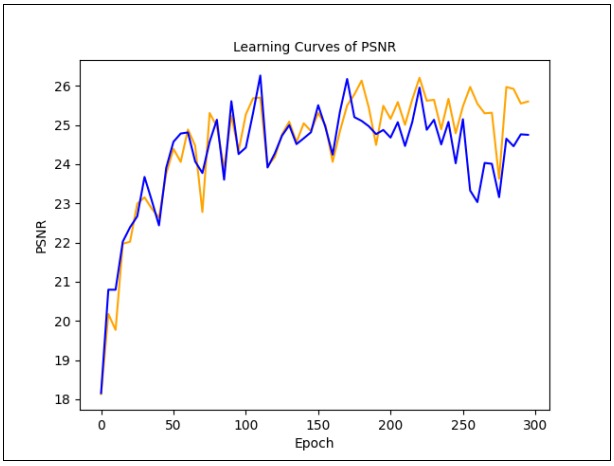


Figure 6 PSNR learning curve

Discuss the results according to your setting of teacher forcing ratio, KL weight, and learning rate.

Table 3 顯示不同 KL Annealing 設定下，validation 和 test 得到的 PSNR 結果比較。

KL Annealing Type	TFR Decay Epoch	Validation PSNR	Test PSNR
Cyclical	150/300	26.20732	**25.79348**
Monotonic	150/300	**26.26468**	25.14493

Table 3 不同 KL Annealing 設定之下，validation/test PSNR 比較

從表格中可以看出在 Monotonic KL annealing 的設定之下，可以得到 validation PSNR 為 26.26468，略高於 Cyclical KL annealing 設定下的 validation PSNR 26.20732，但差距極小。從 Test PSNR 上可以看出兩種模型的效果相當，Test PSNR 皆超過 25，Cyclical KL annealing 達到 25.79348，略高於 Monotonic KL annealing 的 25.14493。

在本次任務使用 KL Annealing 的動機是因為使用 auto-regressive 方法訓練 VAE 經常會面臨 KL vanishing，或者可以稱做 posterior collapse 的問題。意思是在最大化 evidence lower bound (ELBO) 時，模型更關注於降低 reconstruction loss 而不注意 KL loss。當 decoder 能力很強的時候，則模型不會需要 posterior distribution 的幫助，直接從 noise N(0,I) 當中採樣就可以得到很低的 reconstruction loss。這樣造成 input X 和 latent Z 變得互相獨立，decoder 不再依賴 latent Z 進行預測，並且導致 encoded distribution 無法代表真實的資料分布。又或者因為 KL term 在訓練初期約束力過強，導致 posterior distribution 很快退化到高斯分布的 prior distribution，無法提供 decoder 更多信息，這也是造成 KL vanishing 的原因。因此會需要使用 KL Annealing 機制，在訓練初期將 KL term 在 total loss 中的權重設定為 0，鼓勵模型將 input X 的更多信息 encode 進 latent Z 中。隨著訓練 epoch 增加，再逐漸提高 KL term 的權重。

從 Table 2 可以看出兩種訓練策略讓 loss curve 有所不同。在 Cyclical KL annealing 設定下，訓練初期 KL loss 非常高，然後急遽降到接近為 0。隨著 MSE loss 持續降低，PSNR 開始升高。另外，每次 KL 權重重新初始化為 0 時，KL loss 便會再次上升，形成循環。在 Monotonic KL annealing 設定下，則是訓練初期 MSE loss 較高，KL loss 反而在一開始就接近 0。隨著 epoch 增加，MSE loss 逐漸下降，PSNR 則開始升高。

另外，還觀察到 TFR Decay Epoch 若是設定為 0，在訓練初期便不斷降低 teacher forcing ratio 會讓模型預測能力降低。推測是因為在訓練初期，模型還沒有足夠的能力去準確地預測出 reconstruction loss 很小的圖片，因此用前一個 timestep 輸出較差的預測結果用來當作當下 timestep 的輸入，會使得訓練過程不夠穩定。