Deep Learning and Practice

Lab5 - Conditional VAE For Video Prediction

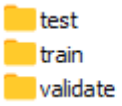
310605009
吳公耀

A. Introduction

1. Lab Objective

- 以 conditional VAE 來對影片作預測，透過前兩張預測接下來的十張
- 實作出 conditional VAE model
- 使用 teacher forcing, KL annealing

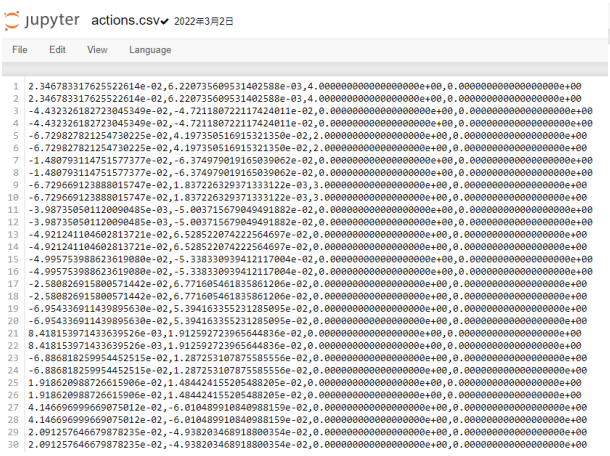
2. Dataset

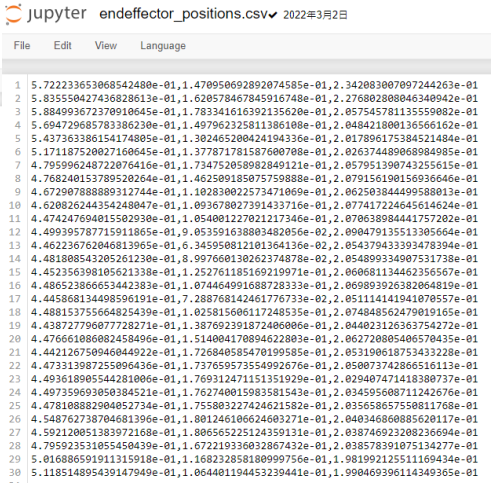



```
train dataset :  43008
validate dataset:  256
```

每個 sequence 裡包含連續的 30 張照片

Condition 為當下 30 張圖片的 action 以及 endeffector_positions





分別為 4 個以及 3 個元素總共 7 個元數

B. Derivation of CVAE (Please use the same notation in Fig.1a)

由 L13，P.23 EM 開始做推導

- To see how the EM works, the chain rule of probability suggests

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

- We next introduce an arbitrary distribution $q(\boldsymbol{Z})$ on both sides and integrate over $\boldsymbol{Z}$

$$\begin{aligned}
&\int q(\boldsymbol{Z}) \log p(\boldsymbol{X};\boldsymbol{\theta}) d\boldsymbol{Z} \\
&= \int q(\boldsymbol{Z}) \log p(\boldsymbol{X},\boldsymbol{Z};\boldsymbol{\theta}) d\boldsymbol{Z} - \int q(\boldsymbol{Z}) \log p(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{\theta}) d\boldsymbol{Z} \\
&= \underbrace{\int q(\boldsymbol{Z}) \log p(\boldsymbol{X},\boldsymbol{Z};\boldsymbol{\theta}) d\boldsymbol{Z} - \int q(\boldsymbol{Z}) \log q(\boldsymbol{Z}) d\boldsymbol{Z}} \\
&+ \underbrace{\int q(\boldsymbol{Z}) \log q(\boldsymbol{Z}) d\boldsymbol{Z} - \int q(\boldsymbol{Z}) \log p(\boldsymbol{Z}|\boldsymbol{X};\boldsymbol{\theta}) d\boldsymbol{Z}}
\end{aligned}$$

to arrive at

VAE 推導：$L = \sum_x \log P(x)$

$$\log P(x) = \int q(z|x) \log p(x)\, dz.$$

$$= \int q(z|x) \log \frac{p(x|z)}{p(z|x)}\, dz$$

$$= \int q(z|x) \log \frac{p(x|z)}{q(z|x)} \frac{q(z|x)}{p(z|x)}\, dz.$$

$$= \int q(z|x) \log \frac{p(x|z)}{q(z|x)}\, dz + \int q(z|x) \log \frac{q(z|x)}{p(z|x)}\, dz.$$

$$= \int q(z|x) \log \frac{p(x|z)}{q(z|x)}\, dz + KL(q(z|x)\,||\,p(z|x)) \geq \int q(z|x) \log \frac{p(x|z)}{q(z|x)}\, dz$$

$$L_{ELBO} = \int q(z|x) \log \frac{p(x|z)p(z)}{q(z|x)}\, dz.$$

$$= \int q(z|x) \log \frac{p(z)}{q(z|x)}\, dz + \int q(z|x) \log p(x|z)\, dz$$

$$= -KL(q(z|x)\,\|\,p(z))\, dz + \int q(z|x) \log p(x|z)\, dz$$

CVAE：

考慮 $p(x|y)$，代入 VAE model

$$\log p(x|y) = \int q(z|x,y) \log \frac{p(x,z|y)}{q(z|x,y)}\, dz + KL(q(z|x,y)\|p(z|x,y))$$

$$L_{ELBO} = \int q(z|x,y) \log \frac{p(x,z|y)}{q(z|x,y)}\, dz$$

$$= \int q(z|x,y) \log \frac{p(x|z,y)p(z|y)}{q(z|x,y)}\, dz$$

$$= \int q(z|x,y) \log \frac{p(z|y)}{q(z|x,y)}\, dz + \int q(z|x,y) \log p(x|z,y)$$

$$= -KL(q(z|x,y)\|p(z|y)) + \int q(z|x,y) \log p(x|z,y)$$

CVAE - 2

$p_\theta(z|y) = p_\theta(z)$

$L(\theta, \phi; x, y) = -KL(q_\phi(z|x,y) \| p_\theta(z)) + E_{q_\phi(z|x,y)} \log p_\theta(x|y,z)$

CVAE - 3

$\log p(y|x)$ 要让 CVAE 引入 x, y 位置

$$\log p(y|x) = \int q(z|x,y) \log \frac{p(y,z|x)}{q(z|x,y)} dz + KL(q(z|x,y) \| p(z|x,y))$$

$$L_{ELBO} = \int q(z|x,y) \log \frac{p(y,z|x)}{q(z|x,y)} dz$$

$$= \int q(z|x,y) \log \frac{p(y|z,x)\, p(z|x)}{q(z|x,y)}$$

$$= \int q(z|x,y) \log \frac{p(z|x)}{q(z|x,y)} dz + \int q(z|x,y) \log p(y|z,x)$$

$$= -KL(q(z|x,y) \| p(z|x)) + \int q(z|x,y) \log p(y|z,x)$$

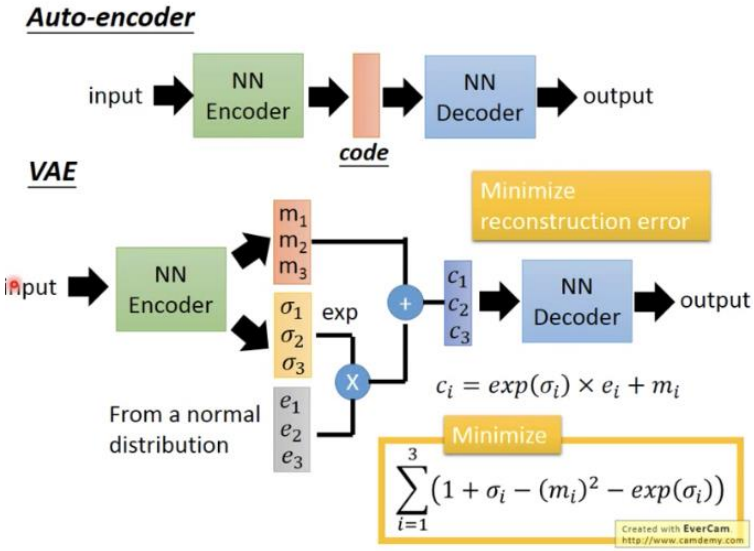
CVAE - 4

假设 $p(x|y,z) = p(x|z)$

$$L_{ELBO} = \int q(z|x,y) \log \frac{p(z|y)}{q(z|x,y)} + \int q(z|x,y) \log p(x|z)$$

C. Implementation details

VAE

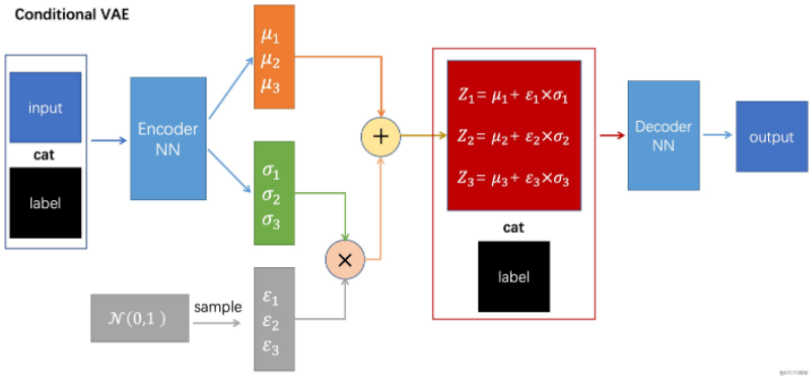


VAE 與 AutoEncoder 不同之處在於 VAE 在編碼過程增加了一些限制，迫使生成的向量遵從高斯分佈。由於高斯分佈可以通過其 mean 和 standard deviation 進行參數化。

VAE 的內部做法：

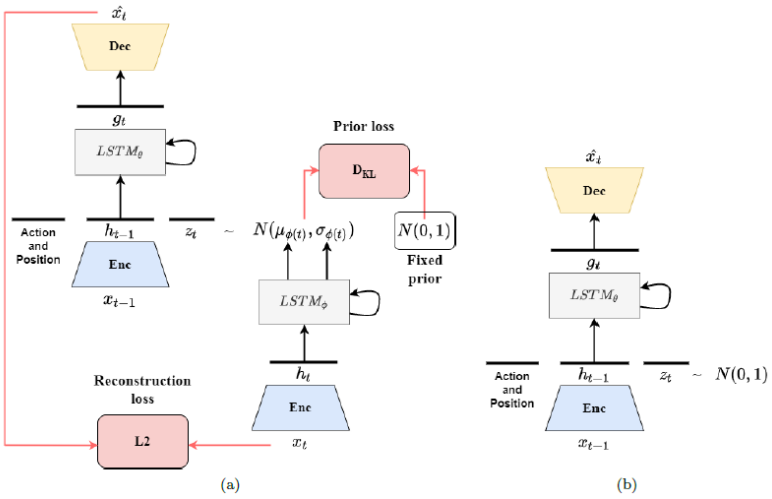
1. 先輸出兩個向量：mean 和 standard deviation
2. 用 normal distribution 產生第三個向量
3. 把第二個向量做 exponential，之後跟第三個向量做相乘後，把它跟第一個向量相加，即成為中間層的隱含向量

CVAE



整體結構和 VAE 差不多，區別是在將數據輸入 encoder 時把 input 與其 label 合併一起輸入，將編碼(Z)輸入 decoder 時把編碼內容與數據 label 合併一起輸入，且 label 並不參與 Loss 計算，CVAE 的 Loss 和 VAE 的 Loss 計算方式相同。生成數據時，可以先從正態分佈採樣，然後 cat 上你想生成的數據的 label，一起送入 Decoder，就能生成和 label 類似的數據。

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



因為助教已經在 sample code 裡把大部分的 VAE 的部分做完了，主要寫的地方就是把 condition 加入原本的 VAE，以及把缺的 reparameterization 部分補上。

輸入的圖片 Xt 以及 Xt-1 frame 經過 sample code 裡面的 vgg encoder 之後，產生 latent vector h_{t-1} 以及 h_t 後分進入 lstm 以及 gaussian_lstm 。

```
class gaussian_lstm(nn.Module):
    def __init__(self, input_size, output_size, hidden_size, n_layers, batch_size, device):
        super(gaussian_lstm, self).__init__()
        self.device = device
        self.input_size = input_size
        self.output_size = output_size
        self.hidden_size = hidden_size
        self.n_layers = n_layers
        self.batch_size = batch_size
        self.embed = nn.Linear(input_size, hidden_size)
        self.lstm = nn.ModuleList([nn.LSTMCell(hidden_size, hidden_size) for i in range(self.n_layers)])
        self.mu_net = nn.Linear(hidden_size, output_size)
        self.logvar_net = nn.Linear(hidden_size, output_size)
        self.hidden = self.init_hidden()

    def init_hidden(self):
        hidden = []
        for _ in range(self.n_layers):
            hidden.append((Variable(torch.zeros(self.batch_size, self.hidden_size).to(self.device)),
                           Variable(torch.zeros(self.batch_size, self.hidden_size).to(self.device))))
        return hidden

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


    def forward(self, input):
        embedded = self.embed(input)
        h_in = embedded
        for i in range(self.n_layers):
            self.hidden[i] = self.lstm[i](h_in, self.hidden[i])
            h_in = self.hidden[i][0]
        mu = self.mu_net(h_in)
        logvar = self.logvar_net(h_in)
        z = self.reparameterize(mu, logvar)
        return z, mu, logvar
```

h_t 透過 gaussian_lstm 中的 mu_net 以及 logvar_net 得到 mu 和 logvar 之後再由 reparameterize 透過將 logvar 做 exponential 然後 normal distribution 產生第三個向量 eps 之後跟第三個向量做相乘後，把它跟第一個向量相加，即成為中間層的隱含向量。

```python
class lstm(nn.Module):
    def __init__(self, input_size, output_size, hidden_size, n_layers, batch_size, device,conditional = True):
        super(lstm, self).__init__()
        self.device = device
        self.conditional = conditional
        if self.conditional:
            self.input_size = input_size +12
            self.hidden_size = hidden_size*2
        else:

            self.input_size = input_size
            self.hidden_size = hidden_size
        self.output_size = output_size

        self.batch_size = batch_size
        self.n_layers = n_layers
        self.embed = nn.Linear(input_size, hidden_size)
        self.embedcond = nn.Linear(7, hidden_size)
        self.lstm = nn.ModuleList([nn.LSTMCell(self.hidden_size, self.hidden_size) for i in range(self.n_layers)])
        self.output = nn.Sequential(
                nn.Linear(self.hidden_size, output_size),
                nn.BatchNorm1d(output_size),
                nn.Tanh())
        self.hidden = self.init_hidden()
    def init_hidden(self):
        hidden = []
        for _ in range(self.n_layers):
            hidden.append((Variable(torch.zeros(self.batch_size, self.hidden_size).to(self.device)),
                           Variable(torch.zeros(self.batch_size, self.hidden_size).to(self.device))))
        return hidden

    def forward(self, input, c=None):

        embedded = self.embed(input)
        embeddedcond = self.embedcond(c)
        embeddedboth = torch.cat((embedded, embeddedcond), dim=1)
        h_in = embeddedboth
        for i in range(self.n_layers):

            self.hidden[i] = self.lstm[i](h_in, self.hidden[i])
            h_in = self.hidden[i][0]

        return self.output(h_in)
```

而目前要預測的 frame 透過 lstm 來將 action 以及 endeffector_positions 組成的 conditional 加入 VAE 模型由條件部分與潛在向量 z 連接起來作為解碼器的輸入也就是 input 為原本的 latent vector 與 prior 計算出的 z 做 concate ， condition 透過過 linear 之後直接跟 input concate。最後再由 sample code 提供的 vgg decoder 解碼產生最後的 predict frame。剛開始有嘗試將 condtion 直接與 input concate 直接進到 LSTM 可是那樣學習的狀況不是很好。

Dataloader 的部分跟之前的作業差不多都是先將 train / validate / test 讀去各自的資料夾裡所有的 sequence 並且存於 dirs，所以__len__也可以直接取裡面的長度。之後再讀去取 sequence 裡面的 frame 並且需要依序從 0 開始拿再將圖片轉成[C, H, W]最後在做 reshape 方便之後轉乘需要的格式。然後再取各個 sequence 中的 condition 並將其 concate 成 7 個元素，最後再依序 index 取 sequence 以及 condition。

```
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.data_dir = ''
        self.istrain = False
        self.d = 0
        self.seq_len = 12
        self.seed_is_set = False
        self.path = ''
        if mode == 'train':
            self.data_dir = 'data/processed_data/train/'
            self.istrain = True
        elif mode == 'test':
            self.data_dir = 'data/processed_data/test/'
        elif mode == 'validate':
            self.data_dir = 'data/processed_data/validate/'
        self.dirs = []
        for dir1 in os.listdir(self.data_dir):
            for dir2 in os.listdir('%s/%s' %(self.data_dir,dir1)):
                self.dirs.append('%s/%s/%s' %(self.data_dir,dir1,dir2))

    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)
```

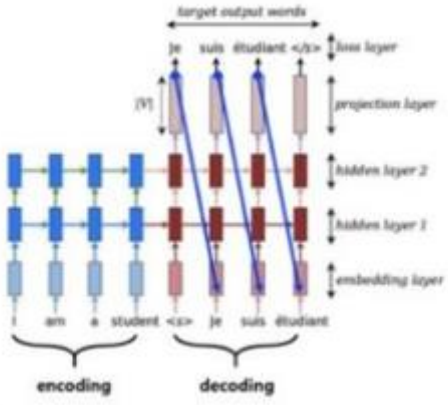
```
def get_seq(self,index):

    d = self.dirs[index]
    self.path = d
    image_seq = []
    for i in range(self.seq_len):
        fname = '%s/%d.png' % (d, i)
        image = Image.open(fname).convert('RGB')
        transform = transforms.Compose([transforms.ToTensor()])
        img = transform(image).reshape(1, 3, 64, 64)
        image_seq.append(img)

    image_seq = np.concatenate(image_seq, axis=0)
    return image_seq.astype(np.float32)


def get_csv(self,index):
    d = self.dirs[index]
    self.path = d
    cond = []
    action_path = '%s/actions.csv' %(self.path)
    action = pd.read_csv(action_path)
    action_tensor = torch.tensor(action.values.astype(np.float32))

    position_path = '%s/endeffector_positions.csv' %(self.path)
    position = pd.read_csv(position_path)
    position_tensor = torch.tensor(position.values.astype(np.float32))

    cond = torch.cat((action_tensor,position_tensor), -1)
    cond_split = torch.split(cond,self.seq_len)
    for i in cond_split:
        return i
    return cond


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

由於輸入的格式為 get_seq: [batch size, frame num, 3, 64, 64]和 get_csv: [batch size, frame num, 7]因此在拿到資料後轉換格式在開始訓練。

```
newseq = seq.permute(1,0,2,3,4)
newcond = cond.permute(1,0,2)
loss, mse, kld, kld_weight= train(newseq, newcond, modules, optimizer, kl_anneal, args, device)
```

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

RNN 存在着兩種訓練模式，1. free-running mode 2. teacher-forcing free-running mode 就是大江上一個 state 的輸入作爲下一個 state 的輸出。而 Teacher Forcing 是一種快速有效地訓練循環神經網絡模型的方法，該模型使用來自先驗時間步長的輸出作爲輸入，訓練的時候不使用上一個 state 的輸出作爲下一個 state 的輸入，而是直接使用訓練數據的標準答案(ground truth)的對應上一項作爲下一個 state 的輸入。



Teacher-Forcing 可以是一個很好的訓練模式是因為:

(1) Teacher-Forcing 能夠在訓練的時候矯正模型的預測，避免在序列生成的過程中誤差進一步放大。
(2) Teacher-Forcing 能夠極大的加快模型的收斂速度，令模型訓練過程更加快&平穩。
(3) Teacher-Forcing 技術是保證 Transformer 模型能夠在訓練過程中完全並行計算所有 token 的關鍵技術。

但相對的也存在著問題

(1) Exposure Bias，也就是訓練和預測的時候 decode 行為的不一致， 導致預測在訓練和預測的時候是從不同的分佈中推斷出來的。而這種不一致導致訓練模型和預測模型直接的 Gap。
(2) Teacher-Forcing 技術在解碼的時候生成的字符都受到了 Ground-Truth 的約束，希望模型生成的結果都必須和參考句一一對應。這種約束在訓練過程中減少模型發散，加快收斂速度。但是一方面也扼殺了翻譯多樣性的可能。
(3) Teacher-Forcing 技術在這種約束下，還會導致一種叫做 Overcorrect(矯枉過正) 的問題。

由於上述的優點以及缺點，這次訓練剛開始用 Teacher-Forcing 避免剛開始訓練往錯誤的方向，之後再透過 non Teacher-Forcing 減少 Teacher-Forcing 可能會產生的問題，且隨著次數增加 non Teacher-Forcing 的訓練模式也更常被使用。

```
if epoch >= args.tfr_start_decay_epoch:
    ### Update teacher forcing ratio ###
    if args.tfr > args.tfr_lower_bound:
        args.tfr -=   args.tfr_decay_step
```

先將所有的 frame 都先 encode 成 h_seq 的 array 後如果是 Teacher-Forcing 則使用 h_seq 裡面的當下 frame 作為 h_{t-1}，反之 non Teacher-Forcing 用 decode 出來的解果作為當下的輸入。之後根據上述的 CVAE 進行訓練

```
for i in range(1, args.n_past + args.n_future):
    if use_teacher_forcing:

        h_target = h_seq[i][0]
        if args.last_frame_skip or i < args.n_past:
            h, skip = h_seq[i-1]
        else:
            h = h_seq[i-1][0]
        z_t, mu, logvar =  modules['posterior'](h_target)
        h = h.to(device)
        z_t = z_t.to(device)
        h_pred = modules['frame_predictor'](torch.cat([h, z_t], 1),cond[i-1])
        x_pred = modules['decoder']([h_pred, skip])

    else:

        h_target = h_seq[i][0]
        if args.last_frame_skip or i < args.n_past:
            h, skip = tmp
        else:
            h = modules['encoder'](tmp)
            h =h[0]
        z_t, mu, logvar =  modules['posterior'](h_target)
        h = h.to(device)
        z_t = z_t.to(device)
        h_pred = modules['frame_predictor'](torch.cat([h, z_t], 1),cond[i-1])
        x_pred = modules['decoder']([h_pred, skip])
        tmp = x_pred
```

最後算出 mse 以及 kld 再透過 kl_annealing 更新 kld 再根據 loss 進行 back propagation，計算 gradient 更新。

```
        mse  += mse_criterion(x_pred,x[i])
        kld += kl_criterion(mu, logvar,args)

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

    optimizer.step()
```

D. Results and discussion

下面的圖表為使用 spec 的 Hyper-parameters and model setting 但是 kl_annealing 分別用 monotonic 以及 cyclical，kl_annealing 都是以 linear 的方式改變。

	monotonic	cyclical
validate	epoch: 300 100%| ave_psnr: 27.368320529134085 best_val_psnr: 28.418320428503307	epoch: 300 100%| it]ave_psnr: 26.975537725035604 best_val_psnr: 28.54475736409921
test	psnr: 26.656277810776807	psnr: 26.8535115524642

1. Show your results of video prediction

(a) Make videos or gif images for test result (select one sequence)

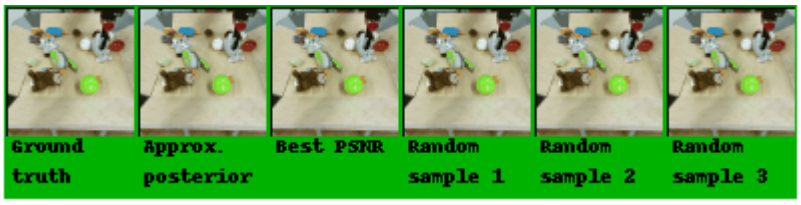
上面的 sequence 為 ground truth 下圖為預測結果

Monotonic(demo_ monotonic.png)

Cyclical(demo_ cyclical.png)

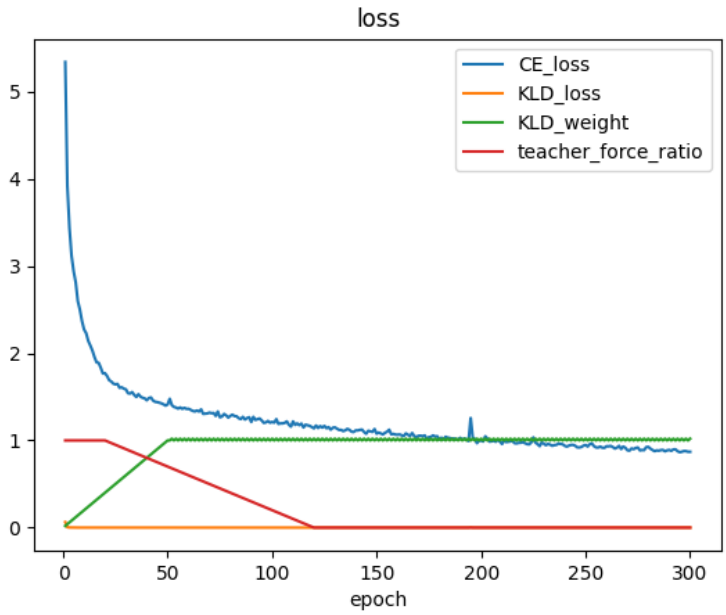
(b) Output the prediction at each time step (select one sequence)

此影片查看檔案 **demo_ monotonic.gif** 以及 **demo_ cyclical.gif**

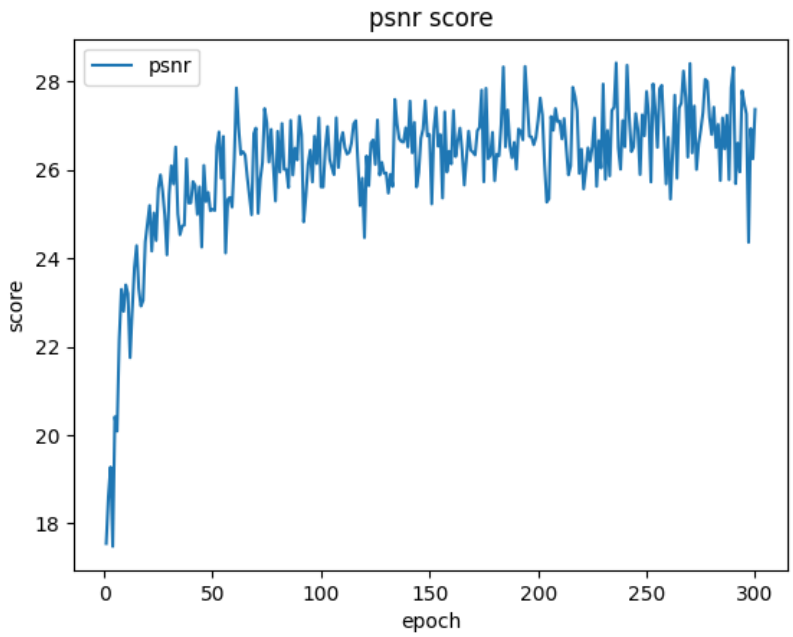


2. Plot the KL loss and PSNR curves during training

Monotonic

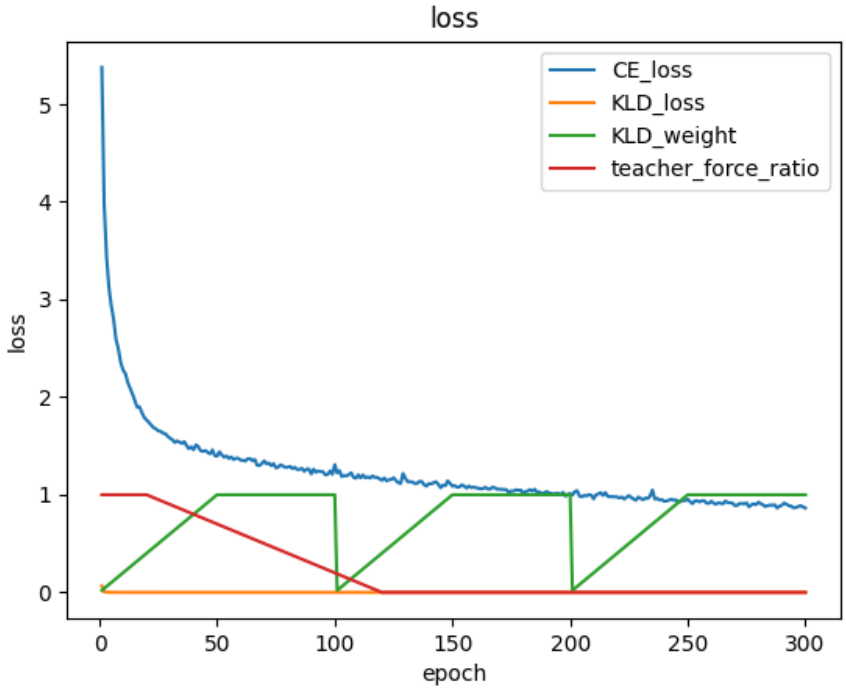


此圖為 Monotonic 以及使用 spec 上的 Hyper-parameter 產生的訓練解果 KL loss、CE_Loss、KL_weight 以及 teacher forcing ratio。KL_weight 上升至 1 就不會再上升，且 teacher forcing ratio 從 epoch 20 開始 -0.01 至 0 為止。可以看出 CE_Loss 也有穩定的在下降，原本以為 KL loss 會像助教的圖片一樣先上升在穩定下降，可是去看 training 過程紀錄好像值都非常小。

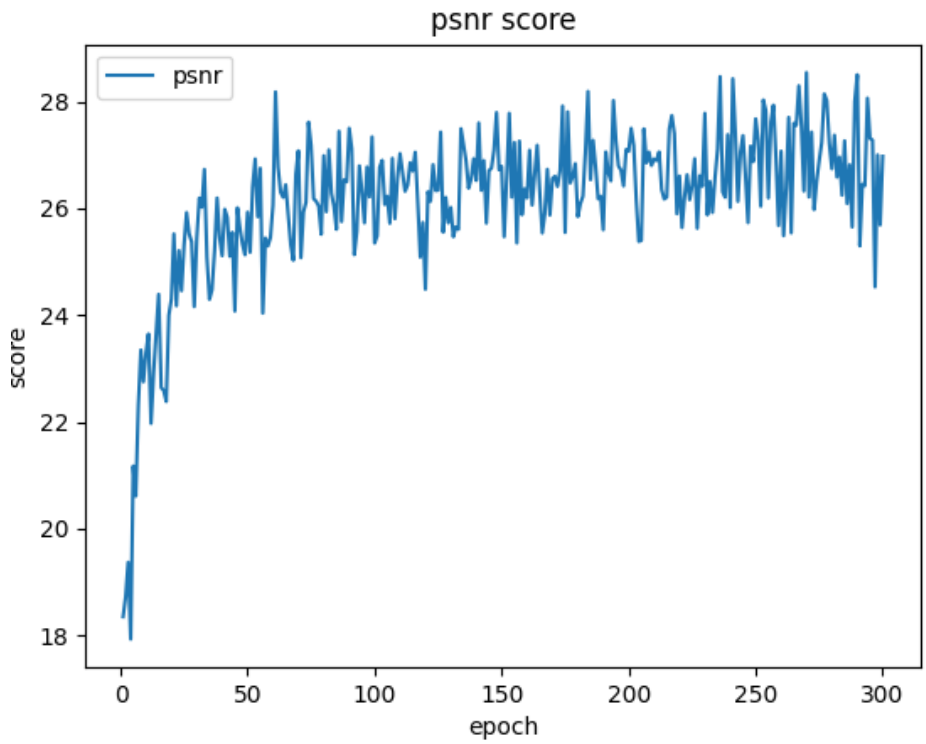


PSNR 解果也有穩定慢慢提升，應該是因為 Monotonic 到了模型訓練較穩定時，KL 都設為 1，讓其有更好的訓練結果

Cyclical



本來以為 cyclicalc 會因為 cycle 變成 0 時，KLD loss 也會隨著 weight 變 0 跟著變大，可是 CE loss 好像比 KLD loss 大太多，所以整個都變成以 CE loss 為主了。而且，收斂的感覺也比 Monotonic 慢。



PSNR 較 Monotonic 來的更動盪，但卻有產生最好 PSNR 得結果。但卻可以解決可能產生 KL 消失的問題

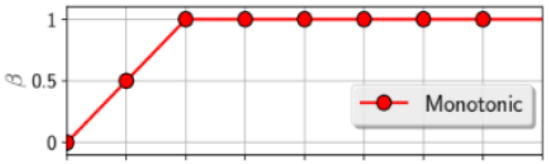
3. Discuss the results according to your setting of teacher forcing ratio, KL weight,and learning rate

Teacher forcing ratio

在 teacher forcing 一開始設置較高，讓模型能盡量避免在錯誤中學習，接著逐漸將他調低，才能夠讓學習更為完整。上面也已經講了 teacher forcing 的重要性了，自己也有跑過訓練的，如果都使用 teacher forcing 沒有回歸原本的預測方式，納到後面不會有很好的進步，反之如果都使用 non teacher forcing 剛開始結果就會沒有那麼理想了。

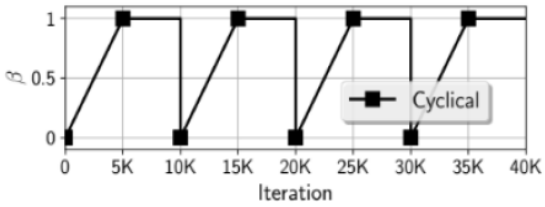
KL weight

Monotonic 方法




```
if self.type == 'monotonic':

    self.v += self.step
    if(self.beta> 1.0):
        self.v =  1.0
    self.beta=self.v
```

Cyclicl 方法




```
else:
    if self.epoch %  self.period ==0:
        self.beta=0.0001

    self.beta += self.step
    if(self.beta> 1.0):
        self.beta =  1.0
```

結果得知，KL 使用 Cyclicl 方法比 Monotonic 方法效果可以得到更好的 PSNR，因為當模型訓練較穩定時，KL 都設為 1，會有更好的訓練結果。但後來在做報告去查才更了解因為：KL 會隨著訓練消失趨近 0 這樣學習到的特徵將不再能夠表達觀測到的數據，也就是透過 Cyclicl 去解決 KL 消失問題，讓訓練中的表現，週期性調節 beta 可以循序漸進地讓結果變好。

learning rate

learning rate 一直以來都是訓練很大的關鍵因素，因為 learning rate 太低，損失函數的變化速度就越慢，容易過擬合。雖然使用低 learning rate 可以確保我們不會錯過任何局部極小值，但也意味著我們將花費更長的時間來進行收斂，特別是在被困在局部最優點的時候。而學習率過高容易發生梯度爆炸，loss 振動幅度較大，模型難以收斂。這次有用比較大的 learning rate 0.01 訓練，但是 spec 給的 0.002 有比較好的解果，也有試過更小的但學習得太慢效果沒有比較好，原本要試試動態調整 learning rate 但是這次訊連平均 300epoch 600 batch size 完成時間需要 2 天左右，所以還沒來的及嘗試。