Conditional VAE for Vido Prediction
Lab Report # 4

By
312581020
許瀚丰

Deep Learning
Spring 2024
Date Submitted: May 7, 2024

Contents

Listings

1 Derivate conditional VAE formula

對於一個conditional的機率分布$p(x|c;\theta)$，我們目標是最大化其的log-likelihood，原式如下

$$\max_{\theta} \int \log p(x|c;\theta)dx \tag{1}$$

假設有一個任意的機率分布$q(z|c)$，我們可以將原式修改為以下

$$\begin{aligned}
\log p(x|c;\theta) &= \int q(z|c)\log p(x|c;\theta)dz \\
&= \int q(z|c)\log(\frac{p(x,z|c;\theta)}{p(z|x,c;\theta)})dz \\
&= \int q(z|c)\log(\frac{p(x,z|c;\theta)}{q(z|c)}\frac{q(z|c)}{p(z|x,c;\theta)})dz \\
&= \int q(z|c)\log(\frac{p(x,z|c;\theta)}{q(z|c)})dz + \int q(z|c)\log(\frac{q(z|c)}{p(z|x,c;\theta)})dz \\
&= \int q(z|c)\log(\frac{p(x,z|c;\theta)}{q(z|c)})dz + KL(q(z|c)||p(z|x,c;\theta))
\end{aligned} \tag{2}$$

由於我們知道對於任意分布p,q，$KL(p||q) \geqslant 0$，因此我們可以得知$\int q(z|c)\log(\frac{p(x,z|c;\theta)}{q(z|c)})dz$必為$\log p(x|c;\theta)$的下界，也就是

$$\log p(x|c;\theta) \geqslant \int q(z|c)\log(\frac{p(x,z|c;\theta)}{q(z|c)})dz \tag{3}$$

我們將右式定義為$L(X,c,q,\theta)$並其展開，可以得到以下

$$\begin{aligned}
L(X,c,q,\theta) &= \int q(z|c)\log(\frac{p(x,z|c;\theta)}{q(z|c)})dz \\
&= \int q(z|c)\log(\frac{p(x|z,c;\theta)p(z|c;\theta)}{q(z|c)})dz \\
&= \int q(z|c)\log p(x|z,c;\theta)dz + \int q(z|c)\log(\frac{(p(z|c;\theta))}{q(z|c)})dz \\
&= \int q(z|c)\log p(x|z,c;\theta)dz - KL(q(z|c)||p(z|c;\theta))
\end{aligned} \tag{4}$$

若此時的$q(z|c)$是由另一個網路ϕ與輸入資料x產生，也就是$q(z|x,c;\phi)$，我們可將原本的結果改寫為以下

$$
\begin{aligned}
&\int q(z|c)\log p(x|z,c;\theta)dz - KL(q(z|c)||p(z|c;\theta)) \\
&= \int q(z|c,x;\phi)\log p(x|z,c;\theta)dz - KL(q(z|c,x;\phi)||p(z|c;\theta)) \\
&= E_{z\sim q(z|x,c;\phi)}\log p(x|z,c;\theta) - KL(q(z|x,c;\phi)||p(z|c;\theta))
\end{aligned} \tag{5}
$$

2 Introduction

2.1 Problem Statement

在本次實驗中，我們需要實作一個cVAE的模型來完成Video Prediction的任務，在訓練時共有兩個輸入，分別為目前人物在上一幀X_{t-1}中畫面的影像與其當前這一幀的Pose影像P_t，而我們需要做的是利用這兩者來去預測出在此時的畫面$\hat{X}_t$。而整體訓練的目標是希望能讓預測結果$\hat{X}_t$與實際的X_t越像越好，並透過的設定不同Teacher Forcing與KL Annealing的方式來幫助訓練，最後會由PSNR作為此次實驗評分的指標。

3 Implementation Details

3.1 How do you write your training protocol

在訓練中，每次輸入進來的shape為(Batch_size, Time_step, Channel, Height, Width)，因此第一個迴圈對於每個Batch，而對於每個Batch的訓練，我將Frame與Pose的組合(X_0, P_1)經由模型所生成$\hat{X}_1$與真實的X_1計算MSELoss，之後則使用預測出的Frame $\hat{X}_1$取代真實的X_1作為新的輸入$(\hat{X}_1, P_2)$並計算Loss，一直持續到$(\hat{X}_{14}, P_{15})$為止。而KLD的部分與前面類似，差別僅在於我們並不需要生成$\hat{X}_0$，因此使用的是(X_1, P_1)到(X_{15}, P_{15})的組合來計算KLD，最終只需要將兩者相加並更新即可。而Teacher Forcing的部分僅需要簡單的將$\hat{X}_i$替換為真實答案X_i，而KL annealing的部分就是直接在一開始先取得beta值，最後再加總Loss Function時將其乘上KLD即可，如程式碼1所示。

```python
def training_one_step(self, img_batch, label_batch, adapt_TeacherForcing):
    # img_batch: (Batch_size, Time_step, Channel, Height, Width) = (2, 16, 3, 32, 64)
    batch_size = img_batch.shape[0] # 2
    total_loss = 0
    beta = self.kl_annealing.get_beta()

    for i in range(batch_size):
        img = img_batch[i] # img: (Time_step, Channel, Height, Width)
        label = label_batch[i] # label: (Time_step, Channel, Height, Width)
        mse = 0
        kld = 0
        x_hat = img[0].unsqueeze(0)

        for j in range(1, img.size(0)):
            if adapt_TeacherForcing:
                prev_img = img[j - 1].unsqueeze(0)
            else:
                prev_img = x_hat

            frame_enc = self.frame_transformation(img[j].unsqueeze(0))
            label_enc = self.label_transformation(label[j].unsqueeze(0))
            z, mu, logvar = self.Gaussian_Predictor(frame_enc, label_enc)
            kld += kl_criterion(mu, logvar, batch_size)

            prev_frame_enc = self.frame_transformation(prev_img).detach()
            decoded = self.Decoder_Fusion(prev_frame_enc, label_enc, z)
            x_hat = self.Generator(decoded)
            mse += self.mse_criterion(x_hat, img[j].unsqueeze(0))
        loss = mse + beta * kld

        self.optim.zero_grad()
        loss.backward()
        self.optimizer_step()

        total_loss += loss

    return total_loss / batch_size
```

Listing 1: Training

3.2 How do you implement reparameterization tricks

我們假設模型輸出結果為μ與$\log \sigma^2$，為何使用$\log \sigma^2$而非直接使用σ是因為後者的輸出要求為非負實數，此項constraint會使模型不好做訓練。若我們要得到σ只需要將$\log \sigma^2$除以2在做exp並即可。

整體reparameterization tricks的部分，我們需要在給定的μ與σ的Normal Distribution中做抽樣，因此我先使用randn_like從$N(0, 1)$抽樣，並將其乘以σ再加上μ就是$N(\mu, \sigma)$了，如程式碼2所示。

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

Listing 2: reparameterization tricks

3.3 How do you set your teacher forcing strategy

要設定Teacher Forcing Ratio(tfr)共有三個參數，分別為tfr, tfr_sde, tfr_d_step，分別代表目前的tfr，多少epoch要降低tfr，每次降低多少tfr。而在我的實作中，就是依照上述的設定去撰寫，每次更新皆會先判斷當前的epoch是否需要更新，而更新時就直接將tfr減去tfr_d_step，需要特別注意不要讓其降到0以下，如程式碼3所示。

```
def teacher_forcing_ratio_update(self):
    if self.current_epoch >= self.tfr_sde and self.current_epoch % self.tfr_sde == 0:
        self.tfr -= self.tfr_d_step
        self.tfr = max(0, self.tfr)
```

Listing 3: teacher forcing

3.4 How do you set your kl annealing ratio

在KL annealing的部分，我直接參考了原始論文的做法，在實驗中我共使用了三種不同的方法，分別為Cyclical, Monotonic, Without(直接設定為1)，且由於在實驗中我發現，由於神經網路較為複雜，因此一開始若完全不使用KL Divergence來更新網路，

可能會導致後續計算Loss產生Nan，因此在一開始我會先會讓計算出的beta不要直接為0來防止Nan的發生。

```python
class kl_annealing():
    def __init__(self, args, current_epoch=0):
        self.annealing_type = args.kl_anneal_type
        assert self.annealing_type in ["Cyclical", "Monotonic", "Without"]
        self.iter = current_epoch + 1

        if self.annealing_type == "Cyclical":
            self.L = self.frange_cycle_linear(num_epoch=args.num_epoch, start=0.0, stop=1.0,  n_cycle=args.kl_anneal_cycle, ratio=args.kl_anneal_ratio)
        elif self.annealing_type == "Monotonic":
            self.L = self.frange_cycle_linear(num_epoch=args.num_epoch, start=0.0, stop=1.0,  n_cycle=1, ratio=args.kl_anneal_ratio)
        else:
            self.L = np.ones(args.num_epoch + 1)

    def update(self):
        self.iter += 1

    def get_beta(self):
        return self.L[self.iter]

    def frange_cycle_linear(self, num_epoch, start=0.0, stop=1.0,  n_cycle=1,ratio=1):
        # adapted from https://github.com/haofuml/cyclical_annealing
        L = np.ones(num_epoch + 1)
        period = num_epoch / n_cycle
        step = (stop - start) / (period * ratio)
        for c in range(n_cycle):
            v , i = start , 0
            while v <= stop and (int(i+c*period) < num_epoch):
                L[int(i + c * period)] = v
                v += step
                i += 1
        return L
```

Listing 4: KL annealing

4 Analysis & Discussion

4.1 Plot Teacher forcing ratio

在本次實驗中，在沒有使用KL anneling的情況(kl-type為Without)下我共測試以下四種不同的Teacher forcing ratio(tfr)設定：

- tfr完全為0：(kl-type_Without_tfr_0.0_teacher-decay_0.1)
- tfr完全為0.1：(kl-type_Without_tfr_0.1_teacher-decay_0.0)
- tfr一開始為0.5，之後每10個epoch下降0.1：(kl-type_Without_tfr_0.5_teacher-decay_0.1)
- tfr一開始為1，之後每5個epoch下降0.5：(kl-type_Without_tfr_1_teacher-decay_0.5)

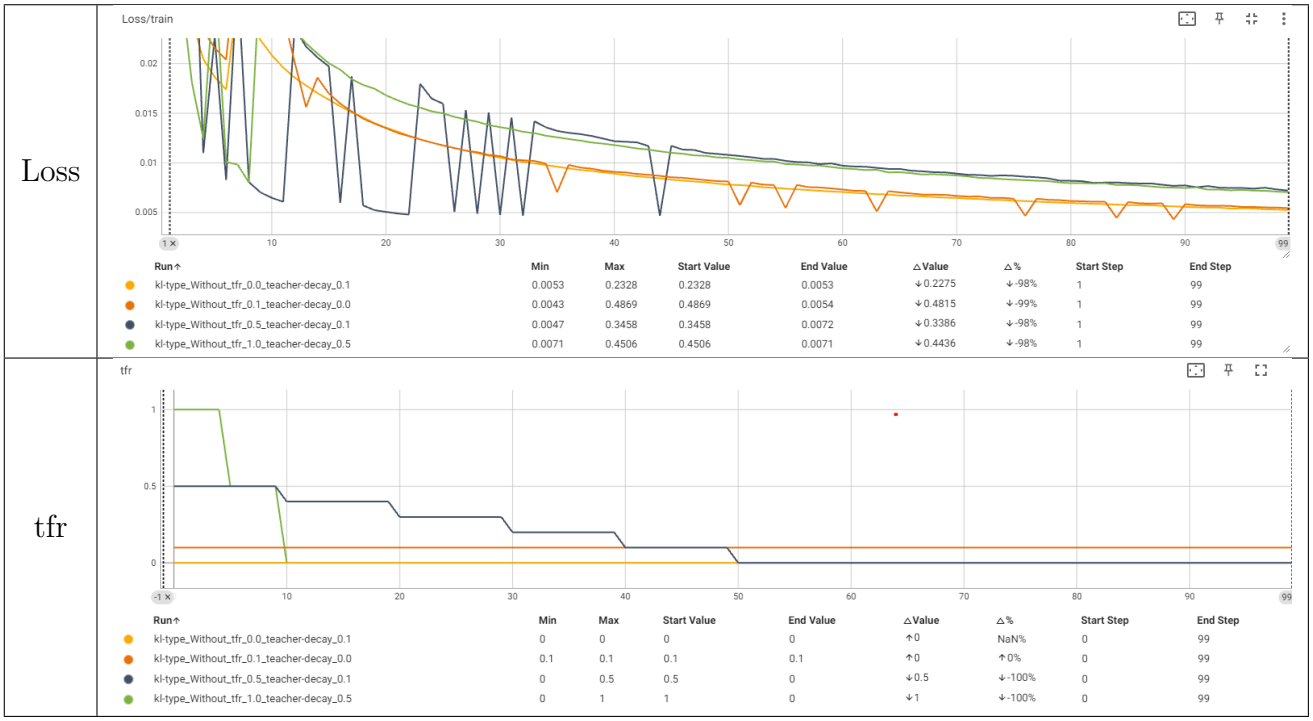


Table 1: Teacher forcing ratio Comparison

實驗結果如表1，從實驗中可以發現，對於本次任務來說，使用Teacher Forcing的效果並不好，甚至容易讓模型提早收斂。我認為其原因可能是因為在Training時，同一個Epoch只會有全部都使用Teacher Forcing或全部都不使用Teacher Forcing，因

此可能會讓模型在過程中陷入local minimum，而在因此在切換時(從使用tfr轉換成不使用tfr, vice versa)就容易產生震盪，進而導致結果受到影響。且由於本次任務的condition(Pose)與要預測的結果(Frame)較為類似，因此儘管不使用Teacher Forcing也能產生出不錯的結果。

4.2 Plot the loss curve while training with different settings

在本次實驗中，在固定Teacher forcing ratio為0的情況下，我共測試以下三種不同的設定：

- KL annealing (Monotonic): kl-type_Monotonic_tfr_0.0_teacher-decay_0.0
- KL annealing (Cyclical): kl-type_Cyclical_tfr_0.0_teacher-decay_0.0
- Without KL Annealing: kl-type_Without_tfr_0.0_teacher-decay_0.0

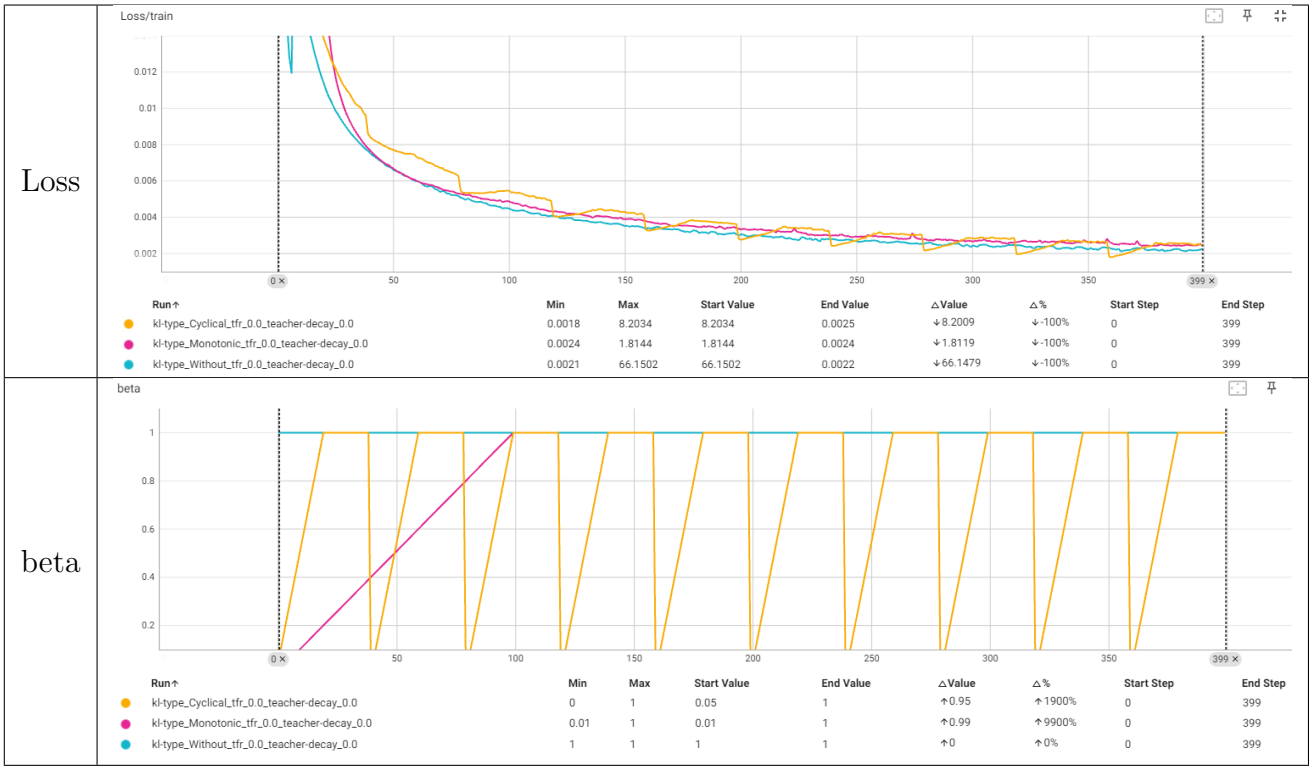


Table 2: KL annealing Comparison

實驗結果如表2，從實驗中可以發現，對於此任務來說，使用不同的KL annealing策略基本上差異不大，但仍可以觀察到使用Cyclical的方式Loss確實會有如設定上的震盪。結束訓練時三種方式的Loss都降至0.002左右，可見三種方式都能有效的讓模型收斂。

4.3 Plot the PSNR-per frame diagram in validation dataset

實驗結果如3，三者皆是固定Teacher forcing ratio為0且訓練400 epoch並比較更改不同KL Annealing所產生出的結果。可以觀察三者的平均PSNR皆有37以上，且每個Frame的PSNR差異並不大。

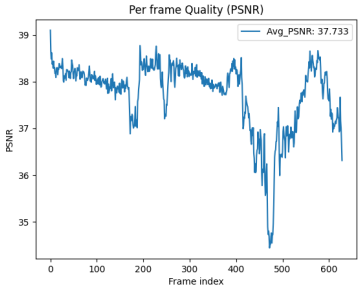
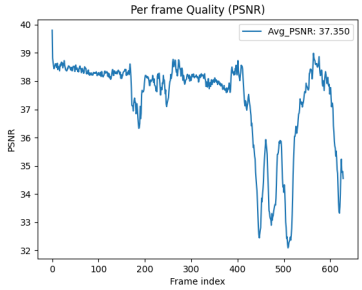
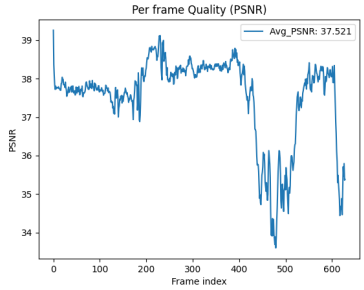
Monotonic	Cyclical	Without
		

Table 3: PSNR

4.4 Other training strategy analysis

4.4.1 Model Modification

我們對圖片做前處理時，會將其normalize到0到1之間，然而在原始程式中，Generator在產生出結果時是透過最後的nn.Conv2d直接產生出結果，因此並不會限制pixel皆在0到1之間，因此我在Generator輸出前加上一個Sigmoid Function，在確保圖片可以限制在0到1之間的同時，也可以讓模型在訓練前期更加穩定，如程式碼5。

```
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()
        )

    def forward(self, input):
        return super().forward(input)
```

Listing 5: Generator