Deep Learning and Practice Lab4

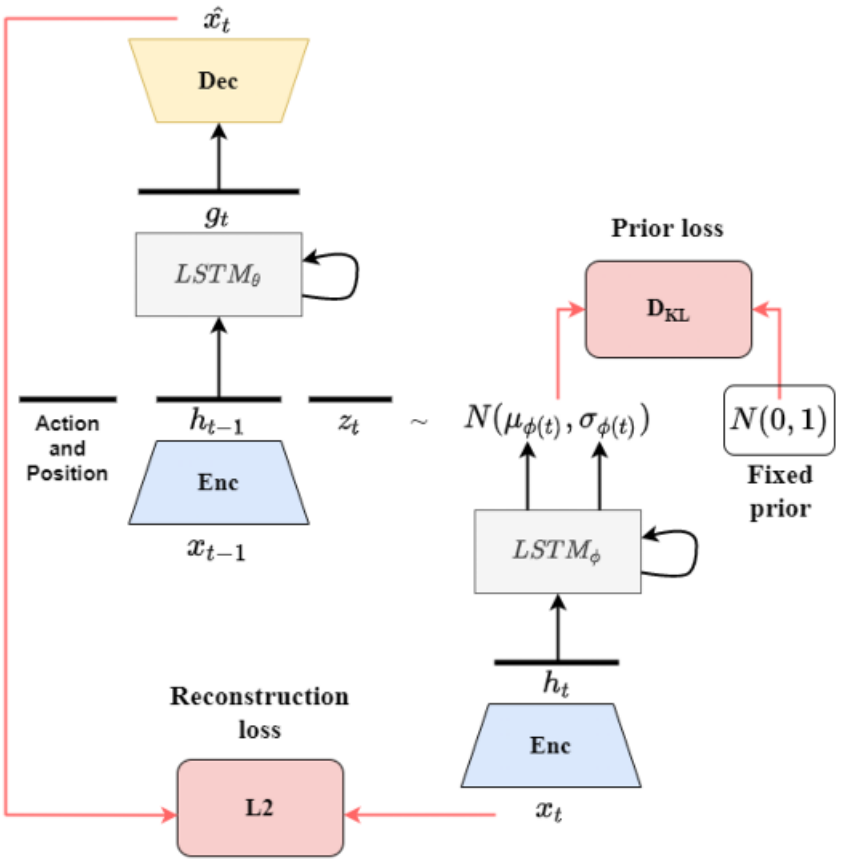
311605011

黃品振

1. Introduction:

這次的作業是要實作一個 cVAE(Conditional variational autoencoder)來實現一個影像預測。我們要預測的是一個機器手臂的運動軌跡，透過給予前面兩張照片生成後面十張預測軌跡運動的圖片。

2. Derivation of CVAE:



(a) Training procedure

上圖是 CVAE 的流程圖，我們的 loss function 是透過兩項相加而成，分別為 KL Divergence 項以及 Reconstruction loss 項。從上圖可以看出訓練流程是我們先輸入 t-1 時刻的圖片經過 Encoder 之後，

將得到 latent code、t-1 時刻的 conditon 資訊，以及 t 時刻的 latent variable 的全部整合在一起輸入進一個 LSTM，再經由 decoder 還原成圖片，此圖片即為我們預測 t 時刻的圖片，我們將此圖片與真正的 t 時刻圖片做對比算出 reconstruction loss，再加上利用 t 時刻真正的圖片經過 LSTM 得到的資料分布與高斯分布 N(0,1)做 KL divergence 的計算，就得到 loss function 了。

3. Implementation details:

A. How I implement my model:

- Dataloader:

```
# --------- load a dataset ------------------------------------
train_data = bair_robot_pushing_dataset(args, 'train')
validate_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)
train_iterator = iter(train_loader)

validate_loader = DataLoader(validate_data,
                          num_workers=args.num_workers,
                          batch_size=args.batch_size,
                          shuffle=True,
                          drop_last=True,
                          pin_memory=True)

validate_iterator = iter(validate_loader)
```

```
test_data = bair_robot_pushing_dataset(args, 'test')
test_loader = DataLoader(test_data,
                         num_workers=args.num_workers,
                         batch_size=args.batch_size,
                         shuffle=True,
                         drop_last=True,
                         pin_memory=True)
test_iterator = iter(test_loader)
```

利用 dataloader，將 dataset.py 裡面的 data 讀進去，分為兩種，有 train 的以及 validate 的資料，還有 test 的資料，然後將 dataloader 讀取的資料套進 iter 的 function 裡面方便迭代。

- Train function:

```
def train(x, cond, modules, optimizer, kl_anneal, args):
    modules['frame_predictor'].zero_grad()
    modules['posterior'].zero_grad()
    modules['encoder'].zero_grad()
    modules['decoder'].zero_grad()
    # initialize the hidden state.
    modules['frame_predictor'].hidden = modules['frame_predictor'].init_hidden()
    modules['posterior'].hidden = modules['posterior'].init_hidden()
    mse = 0
    kld = 0
    x = x.permute(1,0,2,3,4)
    cond = cond.permute(1,0,2)
    use_tfr = True if random.random()< args.tfr else False
    h_seq = [modules['encoder'](x[i]) for i in range(args.n_past + args.n_future)]
    h_notfr = [modules['encoder'](x[0])]
    if  use_tfr:
        for i in range(1, args.n_past + args.n_future):
            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)
            g_pred = modules['frame_predictor'](torch.cat([h, z_t, cond[i-1]], 1))
            x_pred = modules['decoder']([g_pred, skip])
            mse += mse_criterion(x_pred, x[i])
            kld += kl_criterion(mu, logvar, args)
    else:
        for i in range(1, args.n_past + args.n_future):
            h_target = modules['encoder'](x[i])[0]
            if args.last_frame_skip or i < args.n_past:
                h, skip = h_notfr[i-1]
            else:
                h = h_notfr[i-1][0]
            z_t, mu, logvar = modules['posterior'](h_target)
            g_pred = modules['frame_predictor'](torch.cat([h, z_t, cond[i-1]], 1))
            x_pred = modules['decoder']([g_pred, skip])
```

```
            h_notfr.append(modules['encoder'](x_pred))
            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()
    return (loss.detach().cpu().numpy() / (args.n_past + args.n_future),
            mse.detach().cpu().numpy() / (args.n_past + args.n_future),
            kld.detach().cpu().numpy() / (args.n_future + args.n_past),
            beta)
```

這裡可以注意的是在將 x 以及 cond 讀入之後要將第 0 個維度以及第 1 個維度對調，因為原本第 0 個維度的代表的是 batchsize，但是我們在訓練的過程是要讀取圖片，所以要將兩者對調。再來就是在 train function 裡面有用到 teacher forcing 的技巧，teacher forcing 的作法是每一次的 t-1 時刻的輸入都由那個時刻的 ground truth 當作輸入，而不使用 teacher forcing 的方式是將 t-1 上一個時刻生成的圖片經過 encoder 輸入，我們會透過 tfr decay 來讓使用 teacher forcing 的機率越來越低。在 loss 的部分我們會在 KLD 項乘上一個 beta 值，這個 beta 值我們會利用 KL annealing 的方式，訓練剛開始的時候係數大小為 0，給 q(z|x)多一點時間學會把 x 的資訊 encode 到 z 裡，再隨著訓練 step 的增加將係數增大到 1，接著照這個循環去執行我們想要執行的次數。

- KL annealing:

```python
class kl_annealing():
    def __init__(self, args):
        super().__init__()

        iter = args.niter * args.epoch_size
        self.L = np.ones(iter)

        if args.kl_anneal_cyclical:
            cycle = args.kl_anneal_cycle
        else:
            cycle = 1

        period = iter/cycle
        ratio  = args.kl_anneal_ratio
        step = 1.0/(period*ratio)

        for c in range(cycle):
            v, i = 0, 0
            while v <= 1 and (int(i+c*period) < iter):
                self.L[int(i+c*period)] = v
                v += step
                i += 1

        self.beta = 0

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

    def get_beta(self):

        beta = self.L[self.beta]
        self.update()
        return beta
```

上圖即為 kl annealing 的方式，可以透過調整 cycle 的參數來決定要決定要循環幾次。

- Training loop:

```
kl_weight = []
epoch_loss_list = []
ave_psnr_list = []
tfr_list = []
for epoch in range(start_epoch, start_epoch + niter):
    torch.cuda.empty_cache()
    frame_predictor.train()
    posterior.train()
    encoder.train()
    decoder.train()

    epoch_loss = 0
    epoch_mse = 0
    epoch_kld = 0

    for _ in range(args.epoch_size):
        try:
            seq, cond = next(train_iterator)
            #print('seqshape:', seq.shape)
            #print('cond:', cond.shape)
        except StopIteration:
            train_iterator = iter(train_loader)
            seq, cond = next(train_iterator)

        seq = seq.to(device)
        cond = cond.to(device)
        loss, mse, kld, beta= train(seq, cond, modules, optimizer, kl_anneal, args, device)
        epoch_loss += loss
        epoch_mse += mse
        epoch_kld += kld
        a = beta
    epoch_loss_list.append(epoch_loss)#loss_list element:300
    kl_weight.append(a)#kl_weight element:300
```

```
if epoch >= args.tfr_start_decay_epoch:
    ### Update teacher forcing ratio ###
    args.tfr -= args.tfr_decay_step
    if args.tfr < 0:
        args.tfr = 0

tfr_list.append(args.tfr)#tfr_list element:300
```

上圖是 training loop 較為重要的部分，我在每個 epoch 的迭代都將得到的 KL weight(beta)、teacher forcing rate、以及 loss 加入到一個 list 裡面，以方便後續的製圖。而 epoch size 則代表在每一個 epoch 都會迭代 600 筆資料。

- Plot:

```
#plot
tfr_list = np.array(tfr_list)
kl_weight = np.array(kl_weight)
epoch_loss_list = np.array(epoch_loss_list)
xlist = range(0, args.niter, 5)
xlist = list(xlist)
epoch_list = np.arange(args.niter)
#plot loss tfr kl weight
fig, ax1 = plt.subplots()
ax1.set_xlabel('epoch')
ax1.set_ylabel('score/weight')
line1 = ax1.plot(epoch_list, tfr_list, color = 'b', linestyle = "--", label = 'tfr')
line2 = ax1.plot(epoch_list, kl_weight, color = 'g', linestyle = "--", label = 'KL weight')
ax1.tick_params(axis = 'y', labelcolor = 'c')
ax2 = ax1.twinx()
ax2.set_ylabel('Loss')
line3 = ax2.plot(epoch_list, tfr_list, color = 'y', linestyle = "-", label = 'loss')
ax2.tick_params(axis = 'y', labelcolor = 'b')
line = line1 +line2 + line3
labs = [l.get_label() for l in line]
fig.tight_layout()
plt.legend(line, labs, loc = 0)
plt.show()
#plot psnr tfr weight
fig, ax1 = plt.subplots()
ax1.set_xlabel('epoch')
ax1.set_ylabel('score/weight')
ax1.plot(epoch_list, tfr_list, color = 'b', linestyle = "--", label = 'tfr')
ax1.plot(epoch_list, kl_weight, color = 'g', linestyle = "--", label = 'KL weight')
ax1.tick_params(axis = 'y', labelcolor = 'c')
ax2 = ax1.twinx()
ax2.set_ylabel('psnr score')
plt.scatter(xlist, ave_psnr_list, color = 'red')
ax2.tick_params(axis = 'y', labelcolor = 'b')
fig.tight_layout()
fig.legend()
plt.show()
```

透過上圖程式畫出 Loss 圖以及 PSNR 圖。

B. Describe the teacher forcing:

- What is teacher forcing:
  teacher-forcing 在訓練網絡過程中，每次不使用上一個 state 的輸出作為下一個 state 的輸入，而是直接使用訓練數據的標準答案(ground truth)的對應上一項作為下一個 state 的輸入。
- Main idea:
  訓練過程中早期的 RNN 預測能力非常弱，幾乎不能給出好的生成結果。如果某一個 unit 產生了不好的結果，必然會影響後面一片 unit 的學習。teacher forcing 最初的 motivation 就是解決這個問題

的。

- Benefits:
  Teacher forcing 主要是用以改善 free running 難以收斂的問題，利用像是老師帶領著成長的方式訓練。
- Drawbacks:
  teacher-forcing 過於依賴 ground truth 數據，在訓練過程中，模型會有較好的效果，但是在測試的時候因為不能得到 ground truth 的支持，所以如果目前生成的序列在訓練過程中有很大不同，模型就會變得脆弱。

4. Results and discussion:

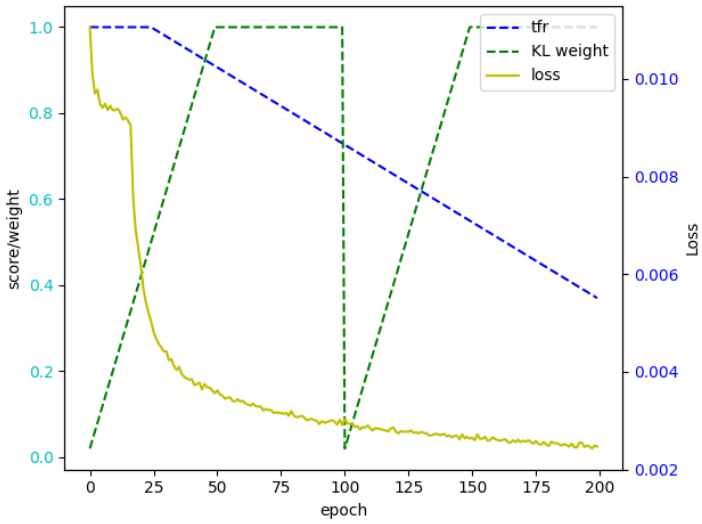
A. Show the results of video prediction:

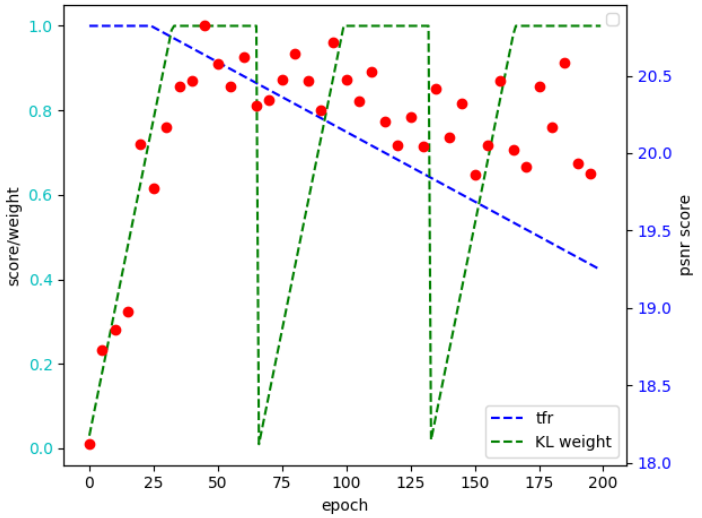
- GIF of test result:

- Output the prediction at each time step:


Ground truth
Approx. posterior
Best SSIM
Random sample 1
Random sample 2
Random sample 3
Ground truth
Approx. posterior
Best SSIM
Random sample 1
Random sample 2
Random sample 3
Ground truth
Approx. posterior
Best SSIM
Random sample 1
Random sample 2
Random sample 3
Ground truth
Approx. posterior
Best SSIM
Random sample 1
Random sample 2
Random sample 3
Ground truth
Approx. posterior
Best SSIM
Random sample 1
Random sample 2
Random sample 3
Ground truth
Approx. posterior
Best SSIM
Random sample 1
Random sample 2
Random sample 3
Ground truth
Approx. posterior
Best SSIM
Random sample 1
Random sample 2
Random sample 3
Ground truth
Approx. posterior
Best SSIM
Random sample 1
Random sample 2
Random sample 3

B. Plot the losses, average PSNR and ratios:





C. Discuss the results according to your settings:

Hyperparameter:

Number of iteration:200

Epoch size:600

Batch size:10

Learning rate:0.002

Optimizer:adam

以上是我的超參數，從所繪製的圖可以發現其實這次的實驗結果並不理想，PSNR 的結果也只有勉強及格而已，問題可能是因為 batch size 不夠大，而且實驗次數不夠多，只有跑幾次而已，報告的結果是跑出來最好的一次，猜測除了 batch size 之外應該程式碼有一些瑕疵。