Lab4 Report

1. Introduction

在這次的 lab 實作了 conditional seq2seq VAE 來做英文單字的時態轉換和生成有 Gaussian noise 的四個時態。

2. Derivation of CVAE

CVAE derivation

$$\log P(X \mid c,\theta) = \log P(X,z \mid c;\theta) - \log p(z \mid X,c;\theta)$$

$$\log p(X \mid c,\theta) = \int q(z \mid X,c;\theta) \log p(X \mid c,\theta)\, dz$$

$$= \int q(z \mid X,c,\theta) \log (X,z \mid c,\theta)\, dz - \int q(z \mid X,c,\theta') \log p(z \mid X,c,\theta)\, dz$$

$$= \int q(z \mid X,c;\theta) \log P(X,z \mid c;\theta)\, dz - \int q(z \mid X,c;\theta') \log P(z \mid X,c;\theta')\, dz$$

$$+ \int q(z \mid X,c,\theta') \log p(z \mid X,c;\theta')\, dz - \int q(z \mid X,c,\theta') \log p(z \mid X,c,\theta)\, dz$$

$$= L(X,q,\theta') - KL(q(z \mid X,c,\theta') \,\|\, p(z \mid X,c;\theta))$$

where $L(X,q,\theta') = \int q(z \mid X,c;\theta') \log P(X,z \mid c;\theta)\, dz - \int q(z \mid X,c,\theta') \log p(z \mid X,c,\theta)\, dz$

$$KL(q(z \mid X,c;\theta') \,\|\, P(z \mid X,c;\theta)) = \int q(z \mid X,c;\theta') \log q(z \mid X,c,\theta')\, dz - \int q(z \mid X,c,\theta') \log p(z \mid X,c,\theta')\, dz$$

$$\Rightarrow L(X,q,\theta') = \log p(X \mid c,\theta) - KL(q(z \mid X,c;\theta') \,\|\, p(z \mid X,c,\theta))$$

$$= E_{z \sim q(z \mid X,c;\theta')} \log p(X \mid z,c;\theta) - KL(q(z \mid X,c;\theta') \,\|\, p(z \mid X,c;\theta))$$

3. Derivation of KL Divergence loss

KL divergence loss derivation in VAE

general KL divergence loss $D_{KL}[P_1 \| P_2] = \frac{1}{2}\left(\log \frac{|\Sigma_2|}{|\Sigma_1|} - n + tr(\Sigma_2^{-1}\Sigma_1) + (\mu_2-\mu_1)^T \Sigma_2^{-1}(\mu_2-\mu_1)\right)$

where $P_1 = N(\mu_1, \Sigma_1)$, $P_2 = N(\mu_2, \Sigma_2)$, in VAE $\Rightarrow$ $P_1 = q(z|x)$, $P_2 = P(z)$

$$\Rightarrow D_{KL}(q(z|x) \| P(z)) = \frac{1}{2}\left(\log \frac{|\Sigma_2|}{|\Sigma_1|} - n + tr(\Sigma_2^{-1}\Sigma_1) + (\mu_2-\mu_1)^T \Sigma_2^{-1}(\mu_2-\mu_1)\right)$$

$$= \frac{1}{2}\left(\log \frac{|I|}{|\Sigma|} - n + tr(\Sigma_2^{-1}\Sigma_1) + (\mu_2-\mu_1)^T \Sigma_2^{-1}(\mu_2-\mu_1)\right)$$

$$= \frac{1}{2}\left(-\log|\Sigma| - n + tr(\Sigma) + \mu^T\mu\right)$$

$$= \frac{1}{2}\left(-\log \prod_i \sigma_i^2 - n + \sum_i \sigma_i^2 + \sum_i \mu_i^2\right)$$

$$= \frac{1}{2}\left(-\sum_i \log\sigma_i^2 - n + \sum_i \sigma_i^2 + \sum_i \mu_i^2\right)$$

$$= \frac{1}{2}\left(-\sum_i (\log\sigma_i^2 + 1) + \sum_i \sigma_i^2 + \sum_i \mu_i^2\right)$$

4. Implementation details

```
(EncoderRNN(
   (condition_embedding): Embedding(4, 8)
   (word_embedding): Embedding(28, 256)
   (gru): GRU(256, 256)
   (mean): Linear(in_features=256, out_features=32, bias=True)
   (logvar): Linear(in_features=256, out_features=32, bias=True)
 ),
 DecoderRNN(
   (latent_to_hidden): Linear(in_features=40, out_features=256, bias=True)
   (word_embedding): Embedding(28, 256)
   (gru): GRU(256, 256)
   (out): Linear(in_features=256, out_features=28, bias=True)
 ))
```

nn.Embedding 將 SOS, a,..., z, EOS 轉成 256 的向量並放入 GRU 中，，然後將 output 的 hidden layer 經過 fc layer 轉成 mean 跟 logvar，再來做 reparameterize 使得 hidden layer 變成 normal distribution，使用 ReLU 函數做激活

Dataloader 的部分使用 np.loadtxt 直接讀取 txt 檔，再對裡面每一行對詞性做分割

reparameterization trick:

```
m = self.mean(hidden)
logvar = self.logvar(hidden)

z = self.sample_z() * torch.exp(logvar/2) + m
```

Loss function: Cross Entropy + KL divergence

KL weight annealing function: Monotonic

```
def KLD_weight_annealing(*args):
    epoch, batch = args
    slope = 0.001
    #slope = 0.1
    scope = (1.0 / slope)*2

    w = (epoch % scope) * slope

    if w > 1.0:
        w = 1.0

    return w
```

Hyperparameters:

```
hidden_size = 256
latent_size = 32
condition_size = 8
teacher_forcing_ratio = 0.5
KLD_weight = 0.0
LR = 0.05
```

Optimizer: SGD

Word generation by Gaussian noise:

```
def generate_word(encoder, decoder, z, condition, maxlen=20):
    encoder.eval()
    decoder.eval()
    z = z.view(1,1,-1)
    sos_token = train_dataset.chardict.word2index['SOS']
    eos_token = train_dataset.chardict.word2index['EOS']
    inputs = torch.LongTensor([sos_token, eos_token])
    outputs = []
    i = 0
    hidden = None

    while True:
        # get (1, word_size)
        output, hidden = decoder(
            inputs.to(device),
            z.to(device),
            encoder.condition(condition),
            False,
            hidden
        )
        output_onehot = torch.max(torch.softmax(output, dim=1), 1)[1]
        if output_onehot.item() == eos_token:
            break

        outputs.append(output_onehot.item())
        i += 1
        if maxlen <= i:
            break

        inputs = torch.LongTensor([outputs[-1], eos_token])

    return torch.LongTensor(outputs)
```

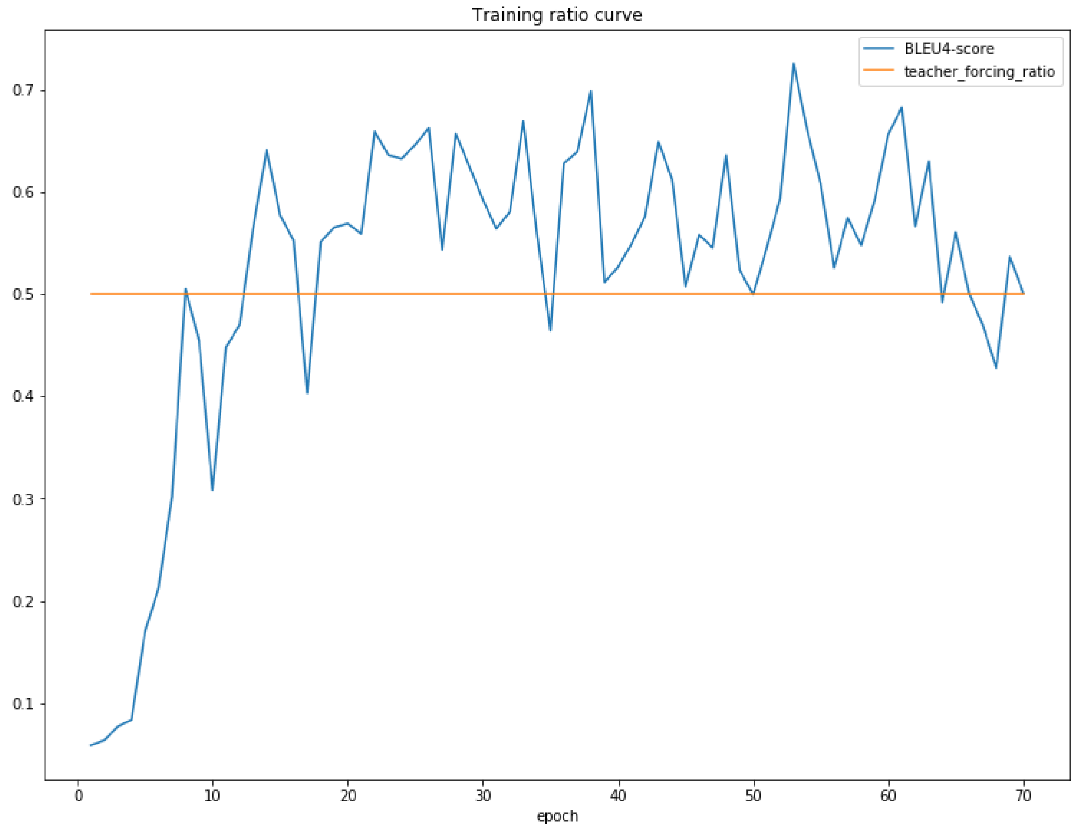
```
def sample_z(self):
    return torch.normal(
        torch.FloatTensor([0]*self.latent_size),
        torch.FloatTensor([1]*self.latent_size)
    ).to(device)
```

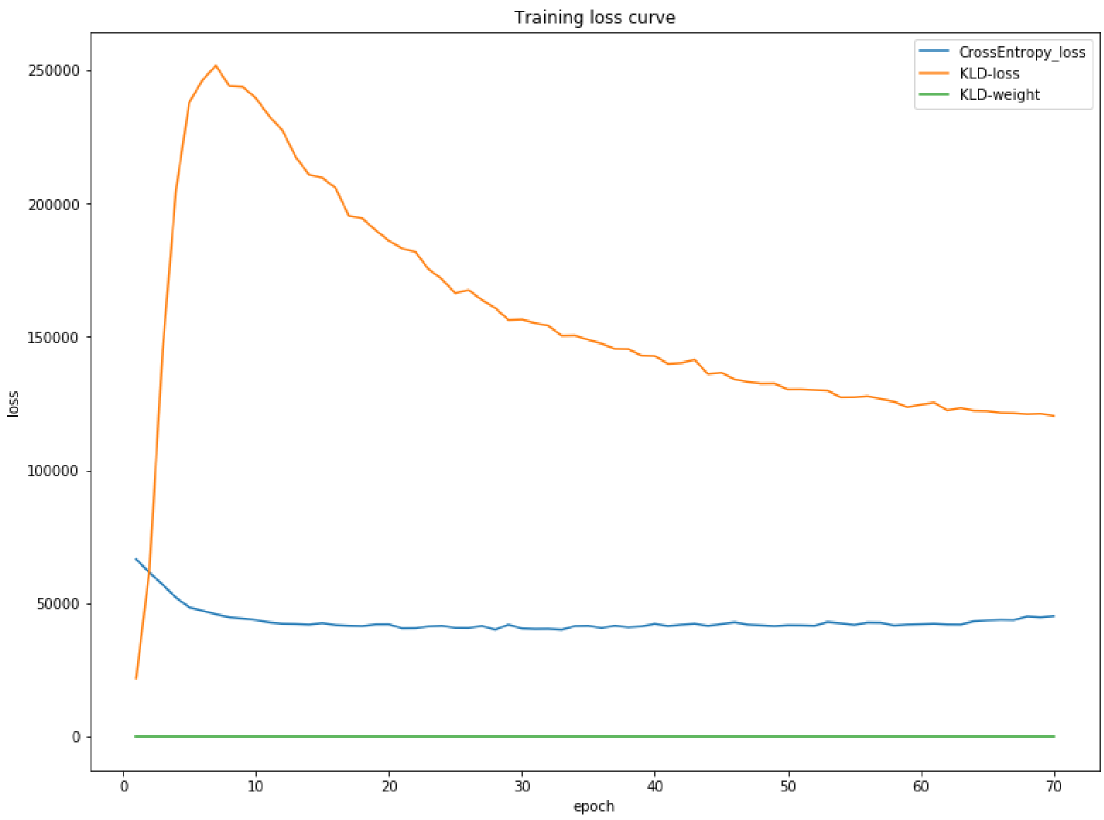
(function in class Encoder)

```
noise = encoder.sample_z()
```

```
outputs = generate_word(encoder, decoder, noise, i)
```

5. Results and discussion





在這次作業中我遇到的最大問題是梯度爆炸，loss 會在幾 10 個 epochs 後因為過大變成 nan，我試過很多方法但效果都不太好，由於我的 model 的 gaussian score 是 0，所以就沒有繪製在圖上，bleu4 score 也可以看出來還沒達到收斂，不是很確定原因，因為我的 kl_weight 一開始有設 0，如果還有時間的話我想再試試。