Deep Learning and Practice Lab4 – VAE

310554009 杜葳葳

1. Introduction

用 CVAE (Conditional Sequence-to-Sequence VAE) 實作「英文動詞時態轉換」，tense 當成 condition，動詞當成 input 和 target。運用 teacher forcing 幫助模型收斂，加上 KL cost annealing，分別運用 Monotonic 和 Cyclical 的策略調整 weight，在 Loss 的 KL 項乘上 weight。最後用 BLEU-4 來評量 test data 的結果，並用 Gaussian normal distribution 來生成四個時態。

2. Derivation of CVAE

很難得到 $P(z|X)$，故用 $q(z)$ 去逼近 $P(z|X)$，因此用 KL-divergence 來計算兩個機率分布的距離，Goal：最小化 $KL(q(z)\|P(z|X))$

$$KL(q(z)\|P(z|X)) = -\sum_z q(z)\log\frac{P(z|X)}{q(z)}$$

$$= -\sum_z q(z)\left[\log\frac{P(X,z)}{q(z)} - \log P(X)\right] \quad \left(P(z|X)=\frac{P(X,z)}{P(X)}\right)$$

$$= -\sum_z q(z)\log\frac{P(X,z)}{q(z)} + \log P(X)$$

$$\Rightarrow \log P(X) = KL(q(z)\|P(z|X)) + \sum_z q(z)\log\frac{P(X,z)}{q(z)}$$

$$= KL(q(z)\|P(z|X)) + L(q)$$

∵ $\log P(X)$ 和 $q(z)$ 無關 ∴ $\log P(X)$ 是固定的

由於 $KL \geq 0$，讓 KL 愈小愈好等同於讓 $L(P)$ 愈大愈好 ⇒ 最大化 $L(P)$ 使 $q(z)$ 接近 $P(z|X)$

$L(P)$ 可當作 marginal log likelihood $\log P(X)$ 的 lower bound，稱為 Evidence Lower Bound (ELBO)

V 和 Expectation 的分佈 $N(0,I)$ 無關

∴ 可以把 Expectation 和 Derivative 交換

$$\nabla_M L(z) = E_{\epsilon\sim N(0,I)}\left[\nabla_M (\log p(x, M+\theta\epsilon) - \log q(M+\theta\epsilon; v))\right]$$

$$\approx \frac{1}{L}\sum_{t=1}^{L} \nabla_M (\log p(x, M+\theta\epsilon) - \log q(M+\theta\epsilon; v)),$$

where $\epsilon \sim N(0,I)$

$$L(v) = E_{z\sim q}[\log p(x,z) - \log q(z;v)]$$

$$= E_{z\sim q}[\log p(x|z) + \log p(z) - \log q(z;v)]$$

$$= E_{z\sim q}[\log p(x|z)] + E_{z\sim q}\left[\log \frac{p(z)}{q(z;v)}\right]$$

$$= E_{z\sim q}[\log p(x|z)] - KL(q(z;v)\|p(z))$$

$$= E_{z\sim q}[\log p(x|z,\theta)] - KL(q(z;v)\|p(z))$$

where $p(x|z)$ is controled by θ

$x \to$ Encoder $\to (M, \theta)$; $p(x|z)$ ↑ Decoder θ

Sample $N(0,I)$ $\to M+\theta\epsilon \to z \to$

ELBO 的 gradient

$$L(v) = E_{z\sim q}[\log p(x,z) - \log q(z;v)]$$

$$\Rightarrow \nabla_v L(v) = \nabla_v \left(E_{z\sim q}[\log p(x,z) - \log q(z;v)]\right)$$

$$g(z,v) = \log p(x,z) - \log q(z;v)$$

What is $\nabla_v L$?

$$\nabla_v L = \nabla_v \int q(z;v)\, g(z,v)\, dz$$

$$= \int \nabla_v q(z;v)\, g(z,v) + q(z;v)\nabla_v g(z,v)\, dz$$

$$= \int q(z;v)\nabla_v \log q(z;v)\, g(z,v) + q(z;v)\nabla_v g(z,v)\, dz$$

$$= E_{q(z;v)}[\nabla_v \log q(z;v)\, g(z,v) + \nabla_v g(z,v)]$$

Using $\nabla_v \log q = \frac{\nabla_v q}{q}$

Reparameterization trick

q 是 Gaussian, $q(z;M,\theta) = N(M,\theta)$, $v = \{M,\theta\}$

$z = M+\theta\epsilon$, $\epsilon \sim N(0,I)$

$$\Rightarrow L(v) = E_{z\sim q}[\log p(x,z) - \log q(z;v)]$$

$$= E_{\epsilon\sim N(0,I)}[\log p(x, M+\theta\epsilon) - \log q(M+\theta\epsilon; v)]$$

3. Derivation of KL Divergence loss

prior : $q_\theta(z) = N(0, I)$

posterior : $p_\theta(z|x^i)$ are gaussian

J is the dimension of z

M : variational mean θ : s.d

$$\int p_\theta(z) \log q_\theta(z) = \int N(z; M, \theta^2) \log N(z; 0, I) dz$$

$$= -\frac{J}{2}\log(2\pi) - \frac{1}{2}\sum_{j=1}^{J}(M_j^2 + \theta_j^2)$$

And $$\int p_\theta(z) \log p_\theta(z) dz = \int N(z; M, \theta^2) \log N(z; M, \theta^2) dz$$

$$= -\frac{J}{2}\log(2\pi) - \frac{1}{2}\sum_{j=1}^{J}(1 + \log \theta_j^2)$$

Therefore:

$$-D_{KL}(p_\theta(z) \| q_\theta(z))$$

$$= \int p_\theta(z)(\log q_\theta(z) - \log p_\theta(z)) dz$$

$$= \frac{1}{2}\sum_{j=1}^{J}(1 + \log((\theta_j)^2) - (M_j)^2 - (\theta_j)^2)$$

4. Implementation details

A. Describe how you implement your model.

a. Dataloader

分別讀入訓練資料與測試資料，並如同 lab3 的步驟，先創建 Dataset 再迭代轉為 Dataloader

```
## train
train_pairs = pd.read_csv('./data/train.txt', header=None).values
train_dataset = TrainDataset(train_pairs)
train_loader = DataLoader(train_dataset, batch_size=1, num_workers=12 ,shuffle=True)

## test
test_pairs = pd.read_csv('./data/test.txt', header=None).values
test_dataset = TestDataset(test_pairs)
test_loader = DataLoader(test_dataset, batch_size=1, shuffle=False)
```

```
class TrainDataset(Dataset):
    def __init__(self, data):
        self.data = []
        self.label = []
        for i in range(len(data)):
            sp, tp, pg, p = data[i][0].split(' ')
            sp = Word2Number(sp)
            tp = Word2Number(tp)
            pg = Word2Number(pg)
            p = Word2Number(p)
            data_pairs = [sp, tp, pg, p]
            for j in range(len(data_pairs)):
                self.data.append(data_pairs[j])
                self.label.append(j)

    def __len__(self):
        return len(self.data)

    def __getitem__(self, index):
        return self.data[index], self.label[index]
```

```
class TestDataset(Dataset):
    def __init__(self, data):
        self.data = []
        self.label = []
        tense = [(0, 3), (0, 2), (0, 1), (0, 1), (3, 1), (0, 2),
                 (3, 0), (2, 0), (2, 3), (2, 1)]
        for i in range(len(data)):
            data1, data2 = data[i][0].split(' ')
            self.data.append([Word2Number(data1), Word2Number(data2)])
            self.label.append(tense[i])

    def __len__(self):
        return len(self.data)

    def __getitem__(self, index):
        return self.data[index], self.label[index]
```

為方便運算，將英文字母變成 ascii 轉為數字，a~z → 2~27、SOS_TOKEN (Start Of Sentence) 是 0、EOS_TOKEN (End Of Sentence) 是 1。

```
fix fixes fixing fixed
[array([ 7, 10, 25,  1]), array([ 7, 10, 25,  6, 20,  1]), array([ 7, 10, 25, 10, 15,  8,  1]), array([ 7, 10, 25,  6,  5,  1])]
```

```python
## 'a'->'2'
def Word2Number(word):
    number = []
    for i in range(len(word)):
        number.append(ord(word[i]) - 97 + 2)
    number.append(EOS_TOKEN) ## EOS_TOKEN: '1'
    return np.array(number)

## '2'->'a'
def Number2Word(number):
    word = ''
    for i in range(len(number)):
        word += chr(number[i] + 97 - 2)
    return word
```

b. Encoder

```python
## encoder initial state
tense = self.tense_embedding(tense).unsqueeze(1) # add one dim
encoder_initial_hidden_state =
    self.encoder.init_hidden_state(self.hidden_size -
    self.condition_embedding_size)
encoder_initial_hidden_state =
    torch.cat([encoder_initial_hidden_state, tense], dim=-1)
encoder_initial_cell_state = self.encoder.init_cell_state()

## encoder
_, hidden_state, cell_state = self.encoder(word,
    encoder_initial_hidden_state, encoder_initial_cell_state)
```

計算 tense 的 embedding 並加上一個維度，初始化 encoder hidden state，把 encoder hidden state 和 tense 接在一起。初始化 cell state (LSTM 用 cell state 來實現 long short memory)

```python
class Encoder(nn.Module):
    def __init__(self, vocab_size, hidden_size):
        super(VAE.Encoder, self).__init__()

        self.hidden_size = hidden_size
        self.embedding = nn.Embedding(vocab_size, hidden_size)
        self.lstm = nn.LSTM(hidden_size, hidden_size)

    def forward(self, x, initial_hidden_state, initial_cell_state):
        word_embedding = self.embedding(x)
        word_embedding = word_embedding.permute(1, 0, 2)
        output, (hidden_state, cell_state) = self.lstm(word_embedding,
            (initial_hidden_state, initial_cell_state))
        return output, hidden_state, cell_state

    def init_hidden_state(self, size):
        return torch.zeros(1, 1, size, device=device)

    def init_cell_state(self):
        return torch.zeros(1, 1, self.hidden_size, device=device)
```

這次作業中的 encoder 要用 nn.LSTM 實作，LSTM 分為 hidden state 和 cell state，hidden state 儲存當前 cell 的運算結果，並傳到現在這一個 cell，而 cell state 則是儲存當前 memory 的值、並傳遞到下一個 cell。

c. Reparameterization Trick

```python
## middle
mean = self.hidden2mean(hidden_state)
variance = self.hidden2variance(hidden_state)
latent = self.reparameterize(mean, variance)
```

```python
self.hidden2mean = nn.Linear(hidden_size, latent_size)
self.hidden2variance = nn.Linear(hidden_size, latent_size)
```

```python
def reparameterize(self, mu, logvar):
    std = torch.exp(0.5 * logvar)
    eps = torch.normal(torch.FloatTensor([0] * self.latent_size),
                       torch.FloatTensor([1] *
                           self.latent_size)).to(device)
    return mu + eps * std
```

使用兩個 Linear layer 將 hidden state 轉為 mean 和 log variance，然後使用 Reparameterization Trick (一種變數代換的方法)，從 N(0, 1)取樣，計算 mean 和 log variance 得到 latent

d. Decoder

```
## decoder initial state
decoder_initial_hidden_state = torch.cat([latent, tense], dim=-1)
decoder_initial_hidden_state =
    self.latent2hidden(decoder_initial_hidden_state)
decoder_initial_cell_state = self.decoder.init_cell_state()

decoder_input = torch.tensor([[SOS_TOKEN]], device=device)
pred_distribution = torch.zeros(word.size(1), self.vocab_size,
    device=device)

## decoder
decoder_hidden_state = decoder_initial_hidden_state
decoder_cell_state = decoder_initial_cell_state
pred_output = []
for i in range(word.size(1)):
    output, decoder_hidden_state, decoder_cell_state =
        self.decoder(decoder_input, decoder_hidden_state,
        decoder_cell_state)
    pred_distribution[i] = output[0]

    if use_teacher_forcing:
        decoder_input = torch.tensor([[word[0][i]]], device=device)
    else:
        if torch.argmax(output).cpu().detach().numpy() == EOS_TOKEN:
            break
        decoder_input = torch.argmax(output).unsqueeze(0).unsqueeze(0)
    pred_output.append(torch.argmax(output).cpu().detach().numpy()
        .item())
```

把 latent 層的 embedding 轉換維度為 decoder 的 hidden state 的維度，並將 [SOS_token] 當成 Decoder 的第一個 input，之後每次 input 一個字元

```
class Decoder(nn.Module):
    def __init__(self, hidden_size, vocab_size):
        super(VAE.Decoder, self).__init__()
        self.hidden_size = hidden_size

        self.embedding = nn.Embedding(vocab_size, hidden_size)
        self.lstm = nn.LSTM(hidden_size, hidden_size)
        self.out = nn.Linear(hidden_size, vocab_size)
        self.softmax = nn.LogSoftmax(dim=1)

    def forward(self, x, hidden_state, cell_state):
        output = self.embedding(x)
        output = F.relu(output)
        output, (hidden_state, cell_state) = self.lstm(output,
            (hidden_state, cell_state))
        output = self.softmax(self.out(output[0]))

        return output, hidden_state, cell_state

    def init_cell_state(self):
        return torch.zeros(1, 1, self.hidden_size, device=device)
```

這次作業中的 decoder 也是使用 nn.LSTM 實作，先將代表字元的數字轉為 embedding，通過 ReLU、softmax 得到 probability，最後判斷是否要使用 teacher forcing，如果要，就拿答案當成下一層 LSTM 的 input，如果不要，就將預測的結果當成 input。

e. KL Cost annealing (adjust KL weight)

為避免 KL vanishing，使用 KL Cost annealing 技巧。

```
def _kl_weight(epoch, method, period):
    if method == 'Monotonic':
        return min(1, epoch / period)
    elif method == 'Cyclical':
        if int(epoch / period) % 2 == 1:
            return 1
        else:
            return (epoch % period)/period
```

- Monotonic：剛開始將 KL weight 設為 0、逐步增大到 1。
- Cyclical：讓 KL weight 隨週期變動。

f. Loss function

Loss 分為兩項，Reconstruction loss (Cross entropy) 和 KL loss 項。

```
ce_loss, kl_loss = loss_function(distribution, word, mean,
    variance, len(output))
loss = ce_loss + kl_weight * kl_loss
```

KL Loss 的計算方式如 3.的證明，得到以下公式：

$$D_{KL}(q_\phi(z) \| p_\theta(z)) = -\frac{1}{2}\sum_{j=1}^{J}(1+\log((\sigma_j)^2)-(\mu_j)^2-(\sigma_j)^2)$$

```
def loss_function(distribution, word, mean, variance, pred_len):
    criterion = nn.CrossEntropyLoss().cuda()

    ## criterion(prediction, target)
    ce_loss = criterion(distribution[:pred_len], word[0][:pred_len])

    kl_loss = -0.5 * torch.sum(1 + variance - mean.pow(2) - variance.exp())
    return ce_loss, kl_loss
```

g. Teacher forcing ratio

Teacher forcing 是在訓練的時候不使用上一個時間點的輸出當成下一個時間點的輸入，而是直接使用標準答案作為下一個時間點的輸入，縮短需要的訓練時間。

```
def _teacher_forcing_ratio(epoch, total_epoch):
    if epoch < (total_epoch/2):
        return 1
    else:
        return 1 - epoch / (total_epoch / 2)
```

最初將 teacher forcing ratio 設為 1，隨 epoch 增加而遞減。

h. Train

```
for epoch in tqdm(range(EPOCHS)):
    model.train()
    gc.collect()
    teacher_forcing_ratio = _teacher_forcing_ratio(epoch, EPOCHS)
    kl_weight = _kl_weight(epoch, KL_METHOD, KL_PERIOD)

    total_ce_loss = 0
    total_kl_loss = 0
    train_total_bleu_4 = 0
    for word, tense in train_loader:
        optimizer.zero_grad()

        word = word.to(device).long()
        tense = tense.to(device).long()

        if random.random() < teacher_forcing_ratio:
            use_teacher_forcing = True
        else:
            use_teacher_forcing = False

        output, distribution, mean, variance = model(word, tense,
            use_teacher_forcing)

        ce_loss, kl_loss = loss_function(distribution, word, mean,
            variance, len(output))
        loss = ce_loss + kl_weight * kl_loss
```

每個 epoch 更新各個參數，並且每次讀入一個單字做訓練、計算 loss。

i. Generate Gaussian

使用 Gaussian noise（而非 encoder 的 output）當成 decoder 的 input。

```
for i in range(100):
    latent = torch.randn(1, 1, LATENT_SIZE).to(device)
    tmp = []
    for i in range(4):
        output = model.generate_gaussian(latent,
            torch.Tensor([i]).to(device).long())
        pred_word = Number2Word(output)
        tmp.append(pred_word)
```

```
def generate_gaussian(self, latent, tense):
    ## encoder initial state
    tense = self.tense_embedding(tense).unsqueeze(1)

    ## decoder initial state
    decoder_initial_hidden_state = torch.cat([latent, tense], dim=-1)
    decoder_initial_hidden_state =
        self.latent2hidden(decoder_initial_hidden_state)
    decoder_initial_cell_state = self.decoder.init_cell_state()

    decoder_input = torch.tensor([[SOS_TOKEN]], device=device)

    decoder_hidden_state = decoder_initial_hidden_state
    decoder_cell_state = decoder_initial_cell_state
    pred_output = []

    while True:
        output, decoder_hidden_state, decoder_cell_state =
            self.decoder(decoder_input, decoder_hidden_state,
            decoder_cell_state)

        if torch.argmax(output).cpu().detach().numpy() == EOS_TOKEN:
            break
        pred_output.append(torch.argmax(output).cpu().detach().numpy()
            .item())
        decoder_input = torch.argmax(output).unsqueeze(0).unsqueeze(0)

    return pred_output
```

B. Specify the hyperparameters

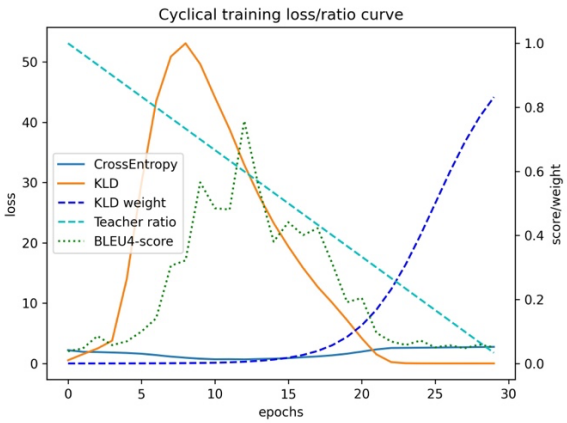
- KL weight：使用 Monotonic 和 Cyclical 兩種調整方式，詳見 Page
- Learning rate：0.05
- Teacher forcing ratio：最初為 1，隨著 epoch 增加、線性降低
- Epochs：30 (有嘗試增加 epochs，但受限於 GPU memory，到 60 多個 epochs 時，會出現 out of memory)

5. Results and discussion

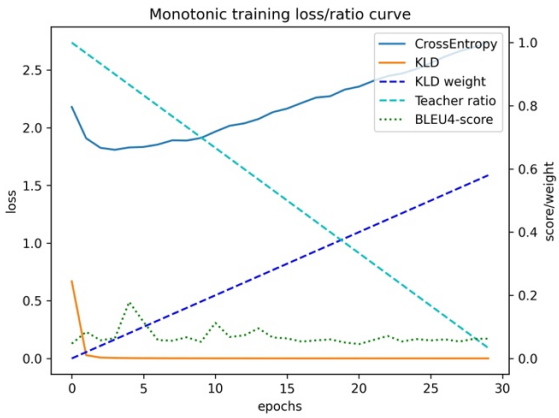
A. Plot the losses, scores and ratios, show results of tense conversion and generation.

a. Plot

Cyclical



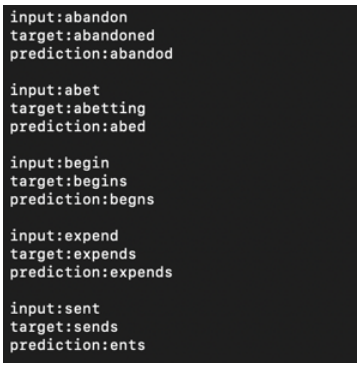
Monotonic

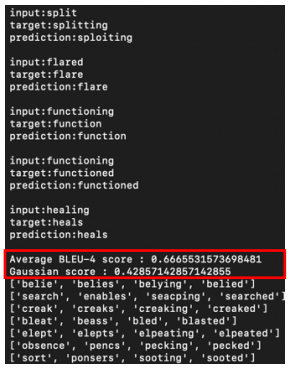


b. Best result: Cyclical

BLEU-4 score: 0.66

Gaussian score: 0.43





B. Discuss the results according to your settings.

嘗試了不同 KL weight 的設計，發現 KL weight 在前面的 epoch 要維持很小、然後再逐漸加大，故後來是使用 sigmoid function 的設計，將一個週期設為 50 個 epochs，Cyclical 普遍比 Monotonic 的表現好。

根據折線圖可以觀察到，當 BLEU score 大的時候，KL weight 低、使得 KL loss 高、Cross entropy loss 低，KL weight 會改變 loss function。

為了要讓 decoder 能用 Gaussian noise 生成不同時態變化，需要降低 KL loss 讓 latent vector 比較接近 Gaussian normal distribution，但這樣就會使得 BLEU score 變低。

嘗試多種 teacher forcing ratio 的設計，1. 初始化為 1，隨 epoch 增加 Linear 遞減，2. 維持 20 個 epoch 為 1，再隨 epoch 數增加逐漸遞減，3. 兩段不同的斜率遞減，但結果相差很有限，故推測 teacher forcing ratio 可能影響不大。

其他：在訓練時，當到達 60 個 epochs 以上會出現 out of memory，故受限於 GPU memory，無法再加大 epochs。