1. Introduction

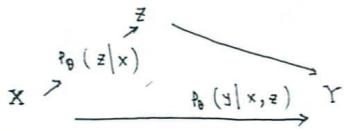
這次的實驗需要實做出 CVAE（條件式變分自編碼器）。VAE 主要的架構由 encoder 和 decoder 組成。encoder 先將原始資料編碼為特定分佈的 latent，decoder 再根據 latent 將資料還原出近似的機率分佈。VAE 只能產生和原始資料類似的輸出資料，而 CVAE 則是將原始資料和其對應的類別共同作為 encoder 的輸入，可以用於生成特定類別的輸出資料。

2. Derivation of CVAE

考慮對 $g^i(z)$ 進行近似推斷。我們可以將 $g^i(z)$ 的範圍限制在某一函式族，例如高斯分佈。那麼可將資料點 x^i 對應之 $g^i(z)$ 記為 $g_\phi(z|x^i)$。

欲對原最大化似然函數 $\ln p(X|\theta)$ 進行優化，使用 reparameterization 方法。先將原機率分佈進行轉換，如將任意常態分佈 $x \sim N(\mu, \sigma)$ 轉 normal 常態分佈 $z \sim N(0,1)$

CVAE 的架構如右



有三種變數 輸入 X，輸出 Y，latent：z，其生成過程為 從 $p(z|x)$ 中取樣 得到 z，再由 $p(y|x,z)$ 生成 輸出 Y。

欲優化 $\ln p(x^i|\theta)$，可先分解 $\ln p(x^i|\theta) = \mathcal{L}(g^i, \theta) + KL(g^i \| p)$，KL 存在下界

$$\log p_\theta = KL(g\|p) - E_{g_\phi(z|x)}[-\log g_\phi(z|x) + \log p_\theta(x,z)]$$

$$\geq -KL(g_\phi(z|x)\|p_\theta(z)) + E_{g_\phi(z|x)}[\log p_\theta(x|z)]$$

$$\mathcal{L}_{VAE}(x;\theta,\phi) = -KL(g_\phi(z|x)\|p_\theta(z)) + \frac{1}{L}\sum_{l=1}^{L} \log p_\theta(x|z^l)$$

$$\log p_\theta(y|x) \geq -KL(g_\phi(z|x,y)\|p_\theta(z|x)) + E_{g_\phi(z|x,y)}[\log p_\theta(y|x,z)]$$

$$\mathcal{L}_{CVAE}(x,y;\theta,\phi) = -KL(g_\phi(z|x,y)\|p_\theta(z|x)) + \frac{1}{L}\sum_{l=1}^{L} \log p_\theta(y|x,z^l)$$

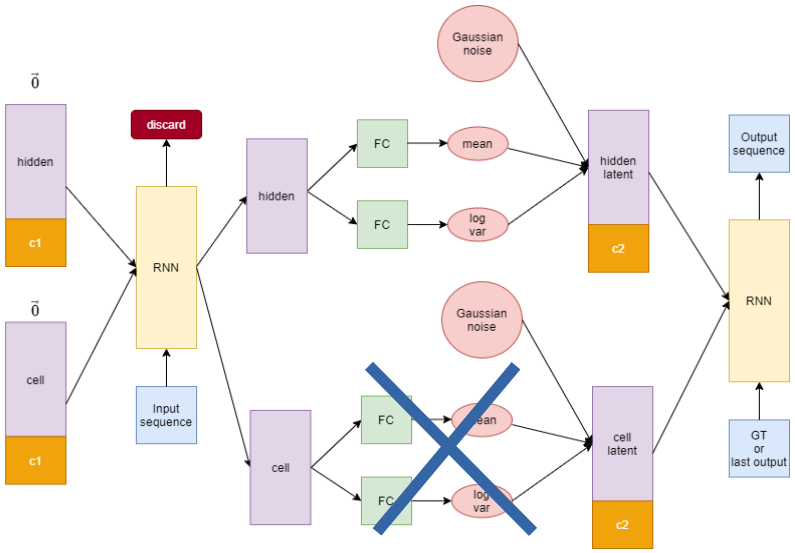
用 Monte Carlo 方法 取樣

$$p_\theta(y|x) \approx \frac{1}{S}\sum_{s=1}^{S} p_\theta(y|x,z^s), \quad z^s \sim p_\theta(z|x)$$

$$p_\theta(y|x) \approx \frac{1}{S}\sum_{s=1}^{S} \frac{p_\theta(y|x,z^s)\, p_\theta(z^s|x)}{g_\phi(z^s|x,y)}, \quad z^s \sim g_\phi(z|x,y)$$

3. Implementation details

將下圖的架構實做出來，除了打叉那一段。其中 RNN 使用 nn.lstm。



(1) encoder

```python
class EncoderRNN(nn.Module):
    def __init__(
        self, word_size, hidden_size, latent_size,
        num_condition, condition_size
    ):
        super(EncoderRNN, self).__init__()
        self.hidden_size = hidden_size
        self.cell_size = hidden_size
        self.condition_size = condition_size
        self.latent_size = latent_size
        self.condition_embedding = nn.Embedding(num_condition, condition_size)
        self.word_embedding = nn.Embedding(word_size, hidden_size)
        self.lstm = nn.LSTM(input_size=hidden_size,hidden_size=hidden_size)
        self.mean = nn.Linear(hidden_size, latent_size)
        self.logvar = nn.Linear(hidden_size, latent_size)
        print('EncoderRNN init done')


    def forward(self, inputs, c, encoder_in_hidden=None, encoder_in_cell=None):
        input_seq = self.word_embedding(inputs.to(device)).view(-1, 1, self.hidden_size)
        c = self.condition(c) #torch.Size([1, 1, 8])
        hidden = torch.zeros(1, 1, self.hidden_size-self.condition_size ,device=device)
        hidden = torch.cat((hidden, c), dim=2)
        cell = torch.zeros(1, 1, self.hidden_size-self.condition_size ,device=device)
        cell = torch.cat((cell, c), dim=2)
        out, (h_n, c_n) = self.lstm(input_seq,(hidden,cell))
        mean = self.mean(h_n) # Latent torch.Size([1, 1, 32])
        logvar = self.logvar(h_n) # Latent torch.Size([1, 1, 32])
        z = (torch.randn(1,1,self.latent_size).to(device)) * torch.exp(logvar/2) + mean # Latent torch.Size([1, 1, 32])
        return out, h_n, c_n, z, mean, logvar # z:latent


    def condition(self, c):
        c = torch.LongTensor([c]).to(device)
        return self.condition_embedding(c).view(1,1,-1)
```

nn.lstm 有三項輸入，原始輸入單字經 word to tensor 轉換後，透過 nn.Embedding 編碼為指定 hidden_size，作為 input sequence 項。
特過 nn.Embedding 將時態編碼成指定 condition_size，再先初始化生成指定 size（hidden_size - condition_size ）的 zeros 接起來，作為 hidden 和 cell 項。

將 nn.lstm 的輸出拉成 linear 後，算出其 mean 和 variance，生成對應的高斯分佈，作為 encoder 的最後輸出，也就是 CVAE 中的 latent 項。

(2) decoder & reparameterization strick

```
class DecoderRNN(nn.Module):
    def __init__(self, word_size, hidden_size, latent_size, condition_size):
        super(DecoderRNN, self).__init__()
        self.hidden_size = hidden_size
        self.word_size = word_size
        self.condition_size = condition_size
        self.latent_to_hidden = nn.Linear(latent_size+condition_size, hidden_size)
        self.word_embedding = nn.Embedding(word_size, hidden_size)
        self.condition_embedding = nn.Embedding(num_condition, condition_size)
        self.lstm = nn.LSTM(input_size=hidden_size,hidden_size=hidden_size)
        self.out = nn.Linear(hidden_size, word_size)
        self.softmax = nn.LogSoftmax(dim=1)
        self.bn = nn.BatchNorm1d(num_features=word_size)
        print('DecoderRNN init done')

    def initCell(self):
        return torch.zeros(1, 1, self.hidden_size - self.condition_size,device=device)

    def forward(self, input_seq, hidden, cell):
        out, (h_n, c_n) = self.lstm(input_seq, (hidden, cell) )
        output = self.out(out[0])
        return output.to(device), out, (h_n, c_n) # output-> prob. distrubution ; out -> latant

    def condition(self, c):
        c = torch.LongTensor([c]).to(device)
        return self.condition_embedding(c).view(1,1,-1)
```

decoder 的操作主要寫在 use_decoder 函式。其前三項輸入，z 為 encoder 的輸出，c 為時態條件，inputs 在 train 過程中為訓練資料（原始單字經 word to tensor）。

使用 decoder 時，一個字母一個字母 decode 並迭代。先將第一個字母 word to tensor（必須為 SOS），經過 nn.Embedding 編碼為 hidden_size，做為第一次的 nn.lstm 之 input sequence 項（圖中寫 GT or last output）。
將 latent_size 的 z 拉長成指定 size（hidden_size - condition_size）後，和透過 nn.Embedding 編碼成指定 condition_size 的 c 接起來，作為第一次的 nn.lstm 之 hidden 項。
初始化生成指定 size（hidden_size - condition_size）的 zeros，和透過 nn.Embedding 編碼成指定 condition_size 的 c 接起來 ，作為第一次的 nn.lstm 之 cell 項。

然後開始迭代，前次 nn.lstm 之輸出 out 為下次 nn.lstm 之 input sequence 項，前次 nn.lstm 之輸出 h_n 為下次 nn.lstm 之 hidden 項，前次 nn.lstm 之輸出 c_n 為下次 nn.lstm 之 cell 項。
直到遇到 out 的機率分佈，最可能結果為 EOS 時，中斷迭代。

另外，若使用 teacher_forcing，則迭代過程中，不使用前次 nn.lstm 之輸出 out，而是將 target 字母經 word to tensor 轉換後，透過 nn.Embedding 編碼為指定 hidden_size，來作為下次 nn.lstm 之 input sequence 項。

```python
def use_decoder(z, c, inputs, use_teacher_forcing):

    sos_token = train_dataset.chardict.word2index['SOS'] # sos_token = 26
    eos_token = train_dataset.chardict.word2index['EOS'] # eos_token = 27
    outputs = []
    if not inputs == None:
        maxlen = inputs.size(0)
    else :
        maxlen = 16

    de_input = torch.LongTensor([sos_token]).to(device)
    de_input_seq = decoder.word_embedding(de_input.to(device)).view(-1, 1, decoder.hidden_size)
    c = decoder.condition(c)
    latent = torch.cat((z, c), dim=2)
    de_hidden = decoder.latent_to_hidden(latent)
    de_cell = decoder.initCell()
    de_cell = torch.cat((de_cell, c), dim=2)

    entropy_loss = 0
    for i in range(1,maxlen):
        output, out, (de_hidden, de_cell) = decoder(de_input_seq, de_hidden, de_cell )
        out_onehot = torch.max(torch.softmax(output, dim=1), 1)[1] # [1]只返回最大值的每个索引
        outputs.append(output)
        if not inputs == None: entropy_loss += criterion(output, torch.tensor([inputs[i]]).to(device))
        if out_onehot.item() == eos_token and not use_teacher_forcing: break
        if use_teacher_forcing:
            de_input = inputs[i]
            de_input_seq = decoder.word_embedding(de_input.to(device)).view(-1, 1, decoder.hidden_size)
        else:
            de_input_seq = out



    if len(outputs) != 0:
        outputs = torch.cat(outputs, dim=0)
    else:
        outputs = torch.FloatTensor([]).view(0, word_size).to(device)

    return outputs, entropy_loss/(maxlen-1)
```

(3) dataloader

在讀取 dataset 時並進行資料的轉換[word to tensor] 和 [時態 to int]。

其中字母 a~z 轉成 0~25，26 為 SOS，27 為 EOS。

時態（現在簡單式，第三人稱，進行式，過去式）分別為（0，1，2，3）。

例如 abandom 轉成 tensor([26， 0， 1， 0，13， 3，14，13，27]),現在簡單式故 c = 0。

```python
class wordsDataset(Dataset):
    def __init__(self, train=True, shuffle=True):
        if train: f = 'dataset/train.txt'
        else: f = 'dataset/test.txt'
        self.datas = np.loadtxt(f, dtype=np.str)

        if train: self.datas = self.datas.reshape(-1)
        else:
            self.targets = np.array([
                [0, 3],  # sp -> p
                [0, 2],  # sp -> pg
                [0, 1],  # sp -> tp
                [0, 1],  # sp -> tp
                [3, 1],  # p  -> tp
                [0, 2],  # sp -> pg
                [3, 0],  # p  -> sp
                [2, 0],  # pg -> sp
                [2, 3],  # pg -> p
                [2, 1],  # pg -> tp
            ])

        self.tenses = [0,1,2,3] #['simple-present','third-person','present-progressive','simple-past']
        self.chardict = CharDict()
        self.train = train

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

    def __getitem__(self, index):
        if self.train:
            c = index % len(self.tenses)
            return self.chardict.longtensorFromString(self.datas[index]), c
        else:
            i = self.chardict.longtensorFromString(self.datas[index, 0])
            ci = self.targets[index, 0]
            o = self.chardict.longtensorFromString(self.datas[index, 1])
            co = self.targets[index, 1]
            return i, ci, o, co
```

```python
class CharDict:
    def __init__(self):
        self.word2index = {}
        self.index2word = {}
        self.n_words = 0
        for i in range(26): self.addWord(chr(ord('a') + i))
        tokens = ["SOS", "EOS"]
        for t in tokens: self.addWord(t)

    def addWord(self, word):
        if word not in self.word2index:
            self.word2index[word] = self.n_words
            self.index2word[self.n_words] = word
            self.n_words += 1

    def longtensorFromString(self, s):
        s = ["SOS"] + list(s) + ["EOS"]
        return torch.LongTensor([self.word2index[ch] for ch in s])

    def stringFromLongtensor(self, l, show_token=False, check_end=True):
        s = ""
        for i in l:
            ch = self.index2word[i.item()]
            if len(ch) > 1:
                if show_token: __ch = "<{}>".format(ch)
                else: __ch = ""
            else: __ch = ch
            s += __ch
            if check_end and ch == "EOS": break
        return s

class wordsDataset(Dataset):
    def __init__(self, train=True, shuffle=True):
        if train: f = 'dataset/train.txt'
```

(4) Guassaion noise 當作 latent，測試 decoder 生成單字的能力

```
words_list = []
for j in range(100):
    words = []
    noise = torch.randn(1,1,latent_size).to(device)
    for i in range(len(train_dataset.tenses)):
        outputs,_ = use_decoder(noise, int(i), None, False)
        outputs_onehot = torch.max(torch.softmax(outputs, dim=1), 1)[1]
        output_str = train_dataset.chardict.stringFromLongtensor(outputs_onehot)
        words.append(output_str)
    words_list.append(words)
gaussian_score = Gaussian_score(words)
if show:
    print('generate word: ')
    for i in range(len(words_list)): print(words_list[i])
    print(' gaussian_score:',Gaussian_score(words_list))
```

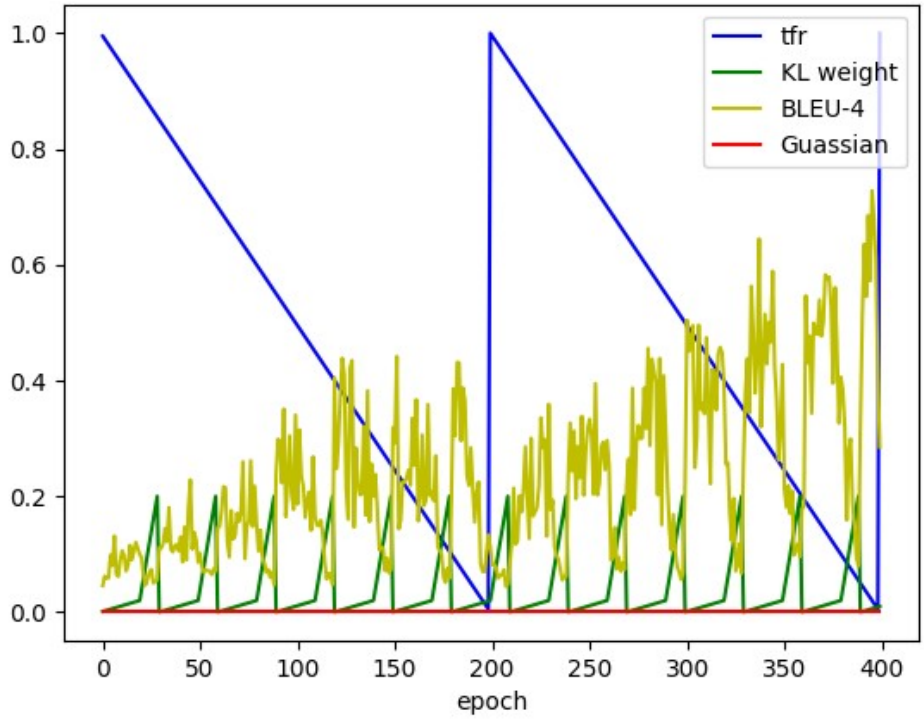
4. Results and discussion

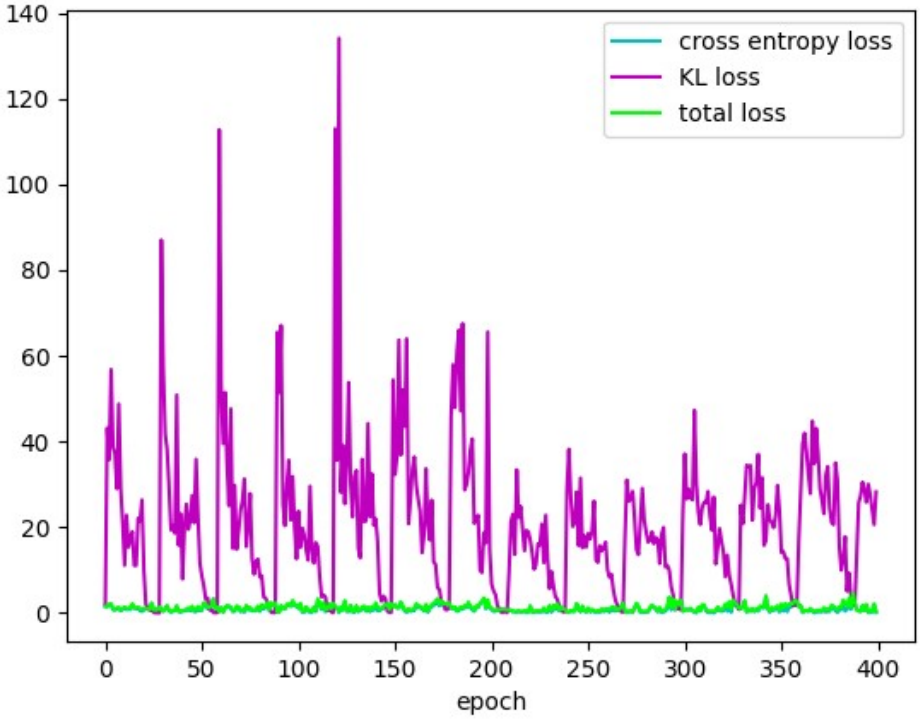
雖然 BLEU 已經達到不錯的分數，但是 Guassian 仍只有 0.02 。

```
input: abandon
target: abandoned
prediction: abandoned

input: abet
target: abetting
prediction: abetting

input: begin
target: begins
prediction: begins

input: expend
target: expends
prediction: expends

input: sent
target: sends
prediction: sends

input: split
target: splitting
prediction: splithing

input: flared
target: flare
prediction: flare

input: functioning
target: function
prediction: finction

input: functioning
target: functioned
prediction: finctioned

input: healing
target: heals
prediction: heals

BLEU-4 score : 0.9086598250425155
```

```
['emplore', 'emplores', 'emploring', 'emplored']
['rrawl', 'rrawls', 'rrawling', 'rrawled']
['crreic', 'crreaces', 'crreiiing', 'crreized']
['chicke', 'chickes', 'cricking', 'cricked']
['rraaxr', 'rrawbss', 'rrawiiing', 'rrapered']
['deel', 'peels', 'peeling', 'peeced']
['curse', 'cursees', 'cursiing', 'curseed']
['demeen', 'demeens', 'demeening', 'deneened']
['casse', 'cassesses', 'cassiniing', 'carsee']
['cuug', 'guugs', 'guuginng', 'guugeed']
['geaver', 'ghavers', 'ghavering', 'ghavered']
['depai', 'deppiss', 'depaining', 'depaid']
['ccnounc', 'acnounces', 'acnouncing', 'ccnounced']
['arnooni', 'arioonizes', 'arnoonizing', 'arnoodized']
['coppese', 'coppesis', 'coppeoiing', 'coppesedn']
['dedoue', 'dedosss', 'dedouing', 'dedoeed']
['despeal', 'mespeals', 'despealing', 'despealed']
['aace', 'aaseclie', 'aaresiing', 'aarese']
['critce', 'crimke', 'cricking', 'crimked']
['maaaen', 'manaens', 'maaaining', 'maaaentd']
['detirt', 'detires', 'detiringg', 'detired']
['grobl', 'growds', 'grobingg', 'grobd']
['creeie', 'creeies', 'creeiing', 'crreiee']
['creahe', 'creehes', 'creahing', 'creahed']
['reea', 'reeales', 'reeeiinn', 'reaaled']
['confin', 'confins', 'condining', 'condined']
['chashen', 'chanhes', 'chanhing', 'chaneed']
['dustert', 'destrrss', 'dustering', 'dusterte']
['switch', 'switches', 'switching', 'switched']
['aspaait', 'adtatins', 'adtatining', 'adtatined']
['curcula', 'curculas', 'curculating', 'curculate']
['firge', 'firges', 'fiiging', 'fiige']
['coaves', 'coavess', 'coaveining', 'coaver']
['cuurrel', 'currrels', 'courruniing', 'currrele']
['dssule', 'dssults', 'dssuling', 'dnsured']
['dosp', 'dossss', 'dosping', 'dosped']
['mesten', 'meshers', 'feshening', 'meshered']
['revei', 'reviises', 'reveiiing', 'reveiee']
['frime', 'srimes', 'sriming', 'frimed']
['indcrtie', 'indertses', 'indertiing', 'indertee']
['create', 'create', 'creating', 'created']
['reaaie', 'craaies', 'rraaiing', 'rraaied']
['demei', 'demeieehes', 'demeiiing', 'demeeee']
['crppo', 'crpsosgs', 'crpposing', 'crpsoned']
['crrs', 'crrssss', 'crrsing', 'crreedd']
['cate', 'dateh', 'catihing', 'cate']
['douse', 'douses', 'dousing', 'doused']
['resri', 'rerries', 'reariigg', 'rearied']
 gaussian_score: 0.02
```

(1) Cyclical





(2) Monotonic

完全 train 不起來，KL weight 恆等於 1 卻無法壓制 KL loss 成長，數值直接爆掉了…

討論：
因為沒有特別撰寫，如何在 loader 中，將每次 batch 中的所有輸入 padding 成一樣長度。所以訓練過程中使用 batch size 為 1。結果遇到了問題，無論輸入是什麼，或是無論 latent 為任何高斯分佈，decoder 吐出來的結果都一樣，而且如果我的 train dataset 中最後一筆為 r 開頭的字，它就只會吐出 r 開頭的字……

```
['reteet', 'reteess', 'reterning', 'reteeted']
['reteet', 'reteess', 'reterning', 'reteeted']
['reteet', 'reteess', 'reterning', 'reteeted']
['reteet', 'reteess', 'reterning', 'reteeted']
['reteet', 'reteess', 'reterning', 'reteeted']
['reteet', 'reteess', 'reterning', 'reteeted']
['reteet', 'reteess', 'reterning', 'reteeted']
['reteet', 'reteess', 'reterning', 'reteeted']
['reteet', 'reteess', 'reterning', 'reteeted']
['reteet', 'reteess', 'reterning', 'reteeted']
['reteet', 'reteess', 'reterning', 'reteeted']
['reteet', 'reteess', 'reterning', 'reteeted']
['reteet', 'reteess', 'reterning', 'reteeted']
['reteet', 'reteess', 'reterning', 'reteeted']
['reteet', 'reteess', 'reterning', 'reteeted']
['reteet', 'reteess', 'reterning', 'reteeted']
['reteet', 'reteess', 'reterning', 'reteeted']
['reteet', 'reteess', 'reterning', 'reteeted']
['reteet', 'reteess', 'reterning', 'reteeted']
['reteet', 'reteess', 'reterning', 'reteeted']
['reteet', 'reteess', 'reterning', 'reteeted']
['reteet', 'reteess', 'reterning', 'reteeted']
['reteet', 'reteess', 'reterning', 'reteeted']
['reteet', 'reteess', 'reterning', 'reteeted']
['reteet', 'reteess', 'reterning', 'reteeted']
['reteet', 'reteess', 'reterning', 'reteeted']
['reteet', 'reteess', 'reterning', 'reteeted']
 gaussian_score: 0.0
```

原因是，整個程式中使用了兩個 model，沒有將他們包成 VAE，故分別用兩個 optimizer。後來發現是因為我兩個 optimizer 用同樣的 learning rate，導致 encoder 進步太慢，來不及學會將輸出近似為標準常態分佈時，decoder 已經進步過快，故每更新一次，就會學到將所有高斯分佈的輸出都變成那樣。
修改使用 encoder LR=1，decoder LR=0.05，並使用 StepRL 逐步調整 encoder 的 LR，可以成功 train 起來。

觀察訓練過程發現，當加重 KL weight，讓總 loss 中 KL loss 的比重增加，會讓全部的輸出瞬間變得幾乎都一樣，因此 BLEU 分數下降。但因為可以讓 encoder 的輸出又更符合標準常態分佈，當再次降低 KL loss 的比重，繼續以 cross entropy loss 主導訓練 decoder 的解碼能力，它能以更高的 BLEU 為起點開始往上進步。反覆如此循環，得到了不錯的訓練結果。從圖中看出，KL weight 的調配主導了整個訓練結果。

KL loss 成長非常快，因此若一開始讓 KL weight 為零，完全不控制它任其瘋狂成長，當切換 KL weight 大於零的瞬間，最佳化過程中容易數值炸裂。

使用 teaching force ratio 可以讓 decoder 在初學階段更容易收斂，如果不使用，或是降低太快，會 train 不起來。但若降低太慢，訓練結果表現也不好。

總之就是一直調整 KL weight 和 teaching force ratio 的設計，直到可以成功訓練出堪用的 CVAE。