NCTU DLP Lab4-Report
Seq2seq Recurrent Network for English Spelling Correction

廖家鴻 0786009
2020/4/28

Outline

- Introduction
- Derivation of BPTT
- Implementation Details
  - A. Dataloader
  - B. Encoder
  - C. Decoder
  - D. Code of evaluation part
- Results and Discussion
  - A. Plot the training loss curve
  - B. BLUE-4 score testing curve
  - C. Results of spelling correction
  - D. Discussion of the results

Introduction

- **Derivation of BPTT**
- **Implementation Details**
  - A. Dataloader
  - B. Encoder
  - C. Decoder
  - D. Code of evaluation part
- **Results and Discussion**
  - A. Plot the training loss curve
  - B. BLEU-4 score testing curve
  - C. Results of spelling correction
  - D. Discussion of the results

首先會將BPTT推導的過程與結果show出來。

在Implementation中，會呈現：

1. Dataloader如從load json檔進來到形成(錯別字,正確字)的token sequence pairs.
2. LSTM based的encoder、decoder實作。
3. Evaluation中如何將decoder的token sequence output轉成word，並與target正確字計算BLEU-4 score

在Result and Discussion中，會呈現：

1. Training setting的參數說明及loss curve的分析。
2. BLEU-4 score curve的分析，與最佳的分數為0.9734
3. 分析spelling correction results
4. Teacher forcing調整心得

Derivation of BPTT-1

題目 & RNN forward equations

In the derivation part, you should compute $\nabla_w L$ step by step with clear notations. You can see more information in slides of recurrent neural network.

$$\boldsymbol{a}^{(t)} = \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)},$$

$$\boldsymbol{h}^{(t)} = \tanh(\boldsymbol{a}^{(t)}),$$

$$\boldsymbol{o}^{(t)} = \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)},$$

$$\hat{\boldsymbol{y}}^{(t)} = \mathrm{softmax}(\boldsymbol{o}^{(t)}).$$

$$p_{\mathrm{model}}(\boldsymbol{y}^{(t)}|\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(t)}) = \prod_i \left(\hat{y}_i^{(t)}\right)^{\mathbf{1}(y_i^{(t)}=1)}$$

$$L^{(t)} = -\log p_{\mathrm{model}}(\boldsymbol{y}^{(t)}|\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(t)})$$

$$L(\{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(t)}\}, \{\boldsymbol{y}^{(1)}, \boldsymbol{y}^{(2)}, \ldots, \boldsymbol{y}^{(t)}\}) = \sum_t L^{(t)}$$

Chain rule:

$$\boldsymbol{X}_{m\times n} \xrightarrow{g(\boldsymbol{X})} \boldsymbol{Y}_{s\times k} \xrightarrow{f(\boldsymbol{Y})} z_{1\times 1}$$

$$\nabla_{\boldsymbol{X}} z = \sum_j \left(\frac{\partial z}{\partial Y_j}\right) \nabla_{\boldsymbol{X}} Y_j,$$

目標先解Loss對W的gradient

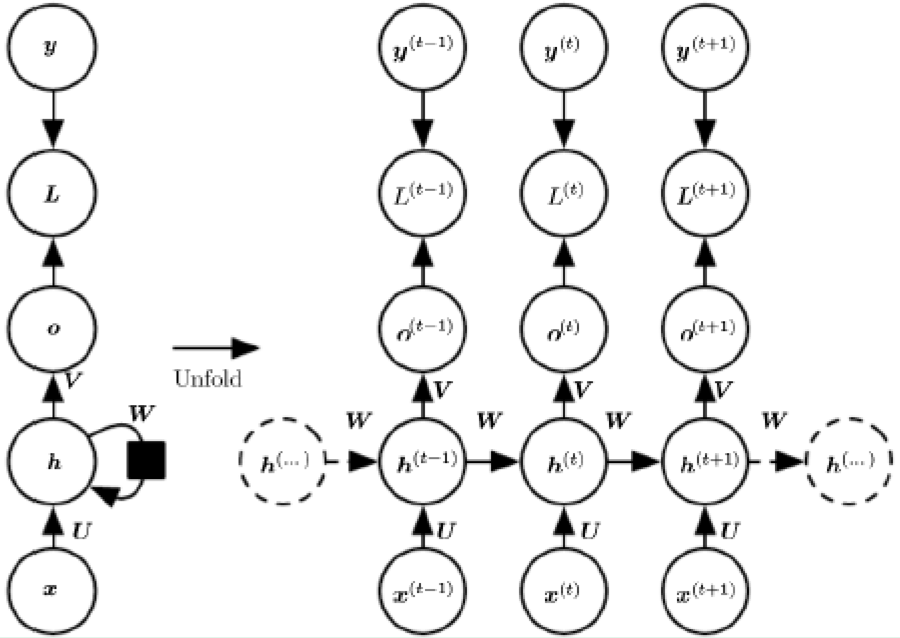
– The immediate child nodes of W are all $h^{(t)}$'s, and

– The chain rule for tensors[a] can be applied to arrive at

$$\nabla_{\boldsymbol{W}} L = \sum_t \sum_i \boxed{\left(\frac{\partial L}{\partial h_i^{(t)}}\right)} (\nabla_{\boldsymbol{W}} h_i^{(t)})$$

從unfold計算圖來看nodes關係並結合chain rule

- An output at each time step with recurrent hidden unit connections



可得 $\boxed{\dfrac{\partial L}{\partial h_i^{(t)}}} = \boxed{\nabla_{h^{(t)}} L = \left(\frac{\partial h^{(t+1)}}{\partial h^{(t)}}\right)^T (\nabla_{h^{(t+1)}} L) + \left(\frac{\partial o^{(t)}}{\partial h^{(t)}}\right)^T (\nabla_{o^{(t)}} L)}$

Derivation of BPTT-2

$$\frac{\partial L}{\partial h_i^{(t)}} = \nabla_{h^{(t)}} L = \left(\boxed{\frac{\partial h^{(t+1)}}{\partial h^{(t)}}}\right)^T (\nabla_{h^{(t+1)}} L) + \left(\frac{\partial o^{(t)}}{\partial h^{(t)}}\right)^T (\nabla_{o^{(t)}} L)$$

由 $\begin{cases} \boxed{\frac{\partial h^{(t+1)}}{\partial h^{(t)}}} = \boxed{\frac{\partial a^{(t+1)}}{\partial h^{(t)}}} \cdot \boxed{\frac{\partial h^{(t+1)}}{\partial a^{(t+1)}}} \\ a^{(t+1)} = b + \boldsymbol{W} h^{(t)} + \boldsymbol{U} x^{(t+1)} \end{cases}$ 可得 $\boxed{\frac{\partial h^{(t+1)}}{\partial h^{(t)}}} = \boxed{\boldsymbol{W}^T} \cdot \boxed{\boldsymbol{H}^{(t+1)}}$

其中 $\boxed{\boldsymbol{H}^{(t+1)}} = \left(\boxed{\frac{\partial h^{(t+1)}}{\partial a^{(t+1)}}}\right)^T$ (H為對角矩陣)

$$= \begin{bmatrix} \boxed{1-(h_1^{(t+1)})^2} & 0 & \cdots & 0 \\ 0 & \boxed{1-(h_2^{(t+1)})^2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \boxed{1-(h_n^{(t+1)})^2} \end{bmatrix}$$

補充 $h^{(t+1)} = \tanh(a^{(t+1)})$

$$\boxed{\frac{\partial h^{(t+1)}}{\partial a^{(t+1)}}} = sech^2(a^{(t+1)}) = 1 - tanh^2(a^{(t+1)}) = \boxed{1-(h^{(t+1)})^2}$$

又 $\begin{cases} (\nabla_{o^{(t)}} L) = \hat{y}^{(t)} - y^{(t)} \\ o^{(t)} = c + Vh^{(t)} \Rightarrow \frac{\partial o^{(t)}}{\partial h^{(t)}} = \boldsymbol{V} \end{cases}$

可得 $\left(\frac{\partial o^{(t)}}{\partial h^{(t)}}\right)^T (\nabla_{o^{(t)}} L) = \boldsymbol{V}^T (\nabla_{o^{(t)}} L)$

因此即可得以下遞迴關係式：

$$\frac{\partial L}{\partial h_i^{(t)}} = \boxed{\nabla_{h^{(t)}} L = \boldsymbol{W}^T \boldsymbol{H}^{(t+1)} (\nabla_{h^{(t+1)}} L) + \boldsymbol{V}^T (\nabla_{o^{(t)}} L)}$$

並且遞迴的最末項為：

$$\nabla_{h^{(\tau)}} L = \boldsymbol{V}^T (\nabla_{o^{(\tau)}} L) = \boldsymbol{V}^T (\hat{y}^{(\tau)} - y^{(\tau)})$$

又 $\nabla_w h^{(t)} = \frac{\partial h^{(t)}}{\partial a^{(t)}} \cdot \frac{\partial a^{(t)}}{\partial w} = \boldsymbol{H}^{(t)} \cdot h^{(t-1)^T}$

$$\nabla_w L = \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}}\right) (\nabla_w h_i^{(t)})$$

$$\boxed{\Rightarrow \nabla_w L = \sum_t \boldsymbol{H}^{(t)} \boxed{(\nabla_{h^{(t)}} L)} h^{(t-1)^T}} \quad \text{\#得解}$$

Derivation of BPTT-3

同理 $\nabla_{\boldsymbol{U}} L = \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}}\right) (\nabla_{\boldsymbol{U}} h_i^{(t)})$

其中 $\frac{\partial L}{\partial h_i^{(t)}}$的推導與$\nabla_{\boldsymbol{w}} L$的過程&結果一樣

又 $\begin{cases} a^{(t)} = b + \boldsymbol{W} h^{(t-1)} + \boldsymbol{U} x^{(t)} \\ \Rightarrow \nabla_{\boldsymbol{U}} h^{(t)} = \frac{\partial h^{(t)}}{\partial a^{(t)}} \cdot \frac{\partial a^{(t)}}{\partial u} = \boldsymbol{H}^{(t)} \cdot {x^{(t)}}^T \end{cases}$

$$\boxed{\therefore \nabla_{\boldsymbol{U}} L = \sum_t \boldsymbol{H}^{(t)} (\nabla_{h^{(t)}} L) {x^{(t)}}^T}$$

同理 $\nabla_{\boldsymbol{b}} L = \sum_t \sum_i \left(\frac{\partial L}{\partial h_i^{(t)}}\right) (\nabla_{\boldsymbol{b}} h_i^{(t)})$

其中 $\frac{\partial L}{\partial h_i^{(t)}}$的推導與$\nabla_{\boldsymbol{w}} L$的過程&結果一樣

又 $\begin{cases} a^{(t)} = b + \boldsymbol{W} h^{(t-1)} + \boldsymbol{U} x^{(t)} \\ \Rightarrow \nabla_{\boldsymbol{b}} h^{(t)} = \frac{\partial h^{(t)}}{\partial a^{(t)}} \cdot \frac{\partial a^{(t)}}{\partial b} = \boldsymbol{H}^{(t)} \cdot 1 \end{cases}$

$$\boxed{\therefore \nabla_{\boldsymbol{b}} L = \sum_t \boldsymbol{H}^{(t)} (\nabla_{h^{(t)}} L)}$$

同理 $\nabla_{\boldsymbol{V}} L = \sum_t \sum_i \left(\frac{\partial L}{\partial o_i^{(t)}}\right) (\nabla_{\boldsymbol{V}} o_i^{(t)})$

又 $\begin{cases} o^{(t)} = c + \boldsymbol{V} h^{(t)} \\ \Rightarrow \nabla_{\boldsymbol{V}} o_i^{(t)} = h^{(t)} \end{cases}$

$$\boxed{\therefore \nabla_{\boldsymbol{V}} L = \sum_t (\nabla_{o^{(t)}} L) {h^{(t)}}^T}$$

同理 $\nabla_{\boldsymbol{c}} L = \sum_t \sum_i \left(\frac{\partial L}{\partial o_i^{(t)}}\right) (\nabla_{\boldsymbol{c}} o_i^{(t)})$

又 $\begin{cases} o^{(t)} = c + \boldsymbol{V} h^{(t)} \\ \Rightarrow \nabla_{\boldsymbol{c}} o_i^{(t)} = 1 \end{cases}$

$$\boxed{\therefore \nabla_{\boldsymbol{c}} L = \sum_t (\nabla_{o^{(t)}} L)}$$

Implementation Details

Implementation Details-Detail of Dataloader-1

```python
def getData(mode):
    if mode == 'train':
        with open('train.json', 'r', encoding='utf-8') as f:
            train_wds = json.load(f)
        inputs = []
        targets = []
        inputs_len = []
        targets_len = []
        for i in range(len(train_wds)):
            for w in train_wds[i]['input']:
                inputs.append(w)
                inputs_len.append(len(w))
                targets.append(train_wds[i]['target'])
                targets_len.append(len(train_wds[i]['target']))
        return np.array(inputs), np.array(targets), np.array(inputs_len), np.array(targets_len)
    else:
        with open('test.json', 'r', encoding='utf-8') as f:
            test_wds = json.load(f)
        inputs = []
        targets = []
        inputs_len = []
        targets_len = []
        for i in range(len(test_wds)):
            for w in test_wds[i]['input']:
                inputs.append(w)
                inputs_len.append(len(w))
                targets.append(test_wds[i]['target'])
                targets_len.append(len(test_wds[i]['target']))
        return np.array(inputs), np.array(targets), np.array(inputs_len), np.array(targets_len)

x_train, y_train, x_train_len, y_train_len = getData('train')
x_test, y_test, x_test_len, y_test_len = getData('test')
word_len = np.hstack([x_train_len, y_train_len, x_test_len, y_test_len])
max_seq_len = int(np.max(word_len) +6)
```

左圖即為將json檔load進來成為numpy array type的code。

我同時也把各word的length算出來，來看所有錯誤與正確的words中，字母最多至幾個，經codes最下方的max_seq_len的搜尋，word最長的有19個字母，然後我故意加6到總sequence長度為25。

因為察看json檔時，發現有多個錯別字對應到同一個正確字，所以用python list append進來時，x_train的總data數即為train.json中的錯別字總數，共有12925筆。

Implementation Details-Detail of Dataloader-1

```
characters = ' '+string.ascii_lowercase

def letter2index(letter):
    return characters.find(letter)

def word2seqToken(line, eos=True):
    ary = np.zeros(len(line))
    tensor = torch.LongTensor(ary)
    eos_tensor = torch.LongTensor([EOS_token])
    for li, letter in enumerate(line):
        tensor[li] = letter2index(letter)
    if eos:
        return torch.cat((tensor,eos_tensor))
    else:
        return tensor

def formPair(x,y):
    wdlist = []
    tglist = []
    for i in range(len(x)):
        w = word2seqToken(x[i])
        t = word2seqToken(y[i])
        wdlist.append(w)
        tglist.append(t)
    return list(zip(wdlist,tglist))
```

```
training_pairs = formPair(x_train,y_train)
random.shuffle(training_pairs)
```

左圖即為將words轉成token sequence的的functions，同時data type轉成torch tensor。並且zip成(錯別字, 正確字)的pair。

所以training_pairs與test_pairs均由formPair()這個function產生。

用random.shuffle是為了在後面training時，每個epoch的input order都隨機打亂而不一樣(我的training設計還是有epochs，後面的slide會進一步解釋)。

另外，我依個人喜好將SOS設成0，EOS設成27，中間的1~26前為那26個小寫字母。Unknow_token則為28(雖然最後我並沒有實作到word dropout的技巧)。

Vocabulary size則設為29，因為有26個字母加上sos、eos、unknown三個token

```
vocab_size = 29
SOS_token = 0
EOS_token = 27
UNK_token = 28
```

Implementation Details-Detail of Encoder

```python
###### Encoder
class EncoderRNN(nn.Module):
    def __init__(self, input_size, hidden_size):
        super(EncoderRNN, self).__init__()
        self.hidden_size = hidden_size
        # self.n_layers = n_layers
        self.embedding = nn.Embedding(input_size, hidden_size)
        # self.gru = nn.GRU(self.hidden_size, self.hidden_size)#, num_layers=
        self.lstm = nn.LSTM(self.hidden_size, self.hidden_size, num_layers=1)
        # self.bilstm = nn.LSTM(hidden_size, hidden_size // 2, num_layers=n_l

    # def forward(self, input, hidden):
    def forward(self, input, hn, cn):
        embedded = self.embedding(input)
        output = embedded.view(1, 1, -1)
        # output, hidden = self.gru(output, hidden)
        output (hn,cn) = self.lstm(output, (hn,cn))
        return output, (hn,cn)
        # return output, hidden#原gru

    def initHidden(self):
        # 各个维度的含义是 (Seguence, minibatch_size, hidden_dim)
        return torch.zeros(1, 1, self.hidden_size, device=device)
```

左圖即LSTM based的encoder class。

與GRU不同的是，LSTM有代表主線的cell stat (cn)，跟代表分線的hidden state (hn)。

Embedding layer在pytorch是吃token值，不是onehot vector，然後會輸出長度為hidden_size的word vector，再餵進lstm裡進行forward。

Implementation Details-Detail of Decoder

```python
##### Decoder
class DecoderRNN(nn.Module):
    def __init__(self, hidden_size, output_size):
        super(DecoderRNN, self).__init__()
        self.hidden_size = hidden_size

        self.embedding = nn.Embedding(output_size, hidden_size)
        # self.gru = nn.GRU(hidden_size, hidden_size)
        self.lstm = nn.LSTM(self.hidden_size, self.hidden_size, num_layers=1)
        self.out = nn.Linear(hidden_size, output_size)
        self.softmax = nn.LogSoftmax(dim=1)

    def forward(self, input, hn, cn):
        output = self.embedding(input).view(1, 1, -1)
        output = F.relu(output)
        # output, hidden = self.gru(output, hidden)
        output, (hn,cn) = self.lstm(output, (hn,cn))
        output = self.out(output[0])
        return output, (hn,cn)

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

左圖即LSTM based的decoder class。

一樣，LSTM有代表主線的cell stat (cn)，跟代表分線的hidden state (hn)

值得一提的是，在training時，decoder與encoder的cell state一定要連接。而另一個hidden state理論上好像不一定要連？但我測試的結果好像都連起來較佳。

```
decoder_hn = encoder_hn
decoder_cn = encoder_cn
```

Implementation Details-Teacher Forcing

```python
decoder_hn = encoder_hn
decoder_cn = encoder_cn
#----------Teacher forcing part----------#
use_teacher_forcing = True if random.random() < Tforce else False

if use_teacher_forcing:
    '''##### Teacher forcing: Feed the target as the next input #####'''
    for di in range(target_length):
        # decoder_output, (decoder_hn,decoder_cn), decoder_attention = decoder(
        #                   decoder_input, decoder_hn,decoder_cn, encoder_outputs)  #encoder_ou
        decoder_output, (decoder_hn,decoder_cn)= decoder(decoder_input, decoder_hn,decoder_cn)

        target_tr = target_tensor[di].unsqueeze(0)  #不unsqueeze的話，dim=None
        loss += criterion(decoder_output, target_tr)
        # loss += criterion(decoder_output, target_tensor[di])
        decoder_input = target_tensor[di]  # Teacher forcing
else:
    '''##### Without teacher forcing: use its own predictions as the next input #####'''
    for di in range(target_length):
        # decoder_output, decoder_hidden, decoder_attention = decoder(
        #                 decoder_input, decoder_hidden, encoder_outputs)
        decoder_output, (decoder_hn,decoder_cn) = decoder(decoder_input, decoder_hn,decoder_cn)
        topv, topi = decoder_output.topk(1)
        decoder_input = topi.squeeze().detach()  # detach from history as input
        target_tr = target_tensor[di].unsqueeze(0)  #不unsqueeze的話，dim=None
        loss += criterion(decoder_output, target_tr)
        # loss += criterion(decoder_output, target_tensor[di])
        if decoder_input.item() == EOS_token:
            break
```

Teacher forcing based的decoder其實跟pytorch的tutorial差不多，主要是實作成沒有attention的時候，要將encoder_outputs拿掉，然後因為是改成LSTM，所以會有(hn,cn)兩個hidden state要加進去。

值得一提的是，在training時，decoder與encoder的cell state一定要連接。而另一個hidden state理論上好像不一定要連？但我測試的結果好像都連起來較佳。

```python
decoder_hn = encoder_hn
decoder_cn = encoder_cn
```

Implementation Details-Code of Evaluation part

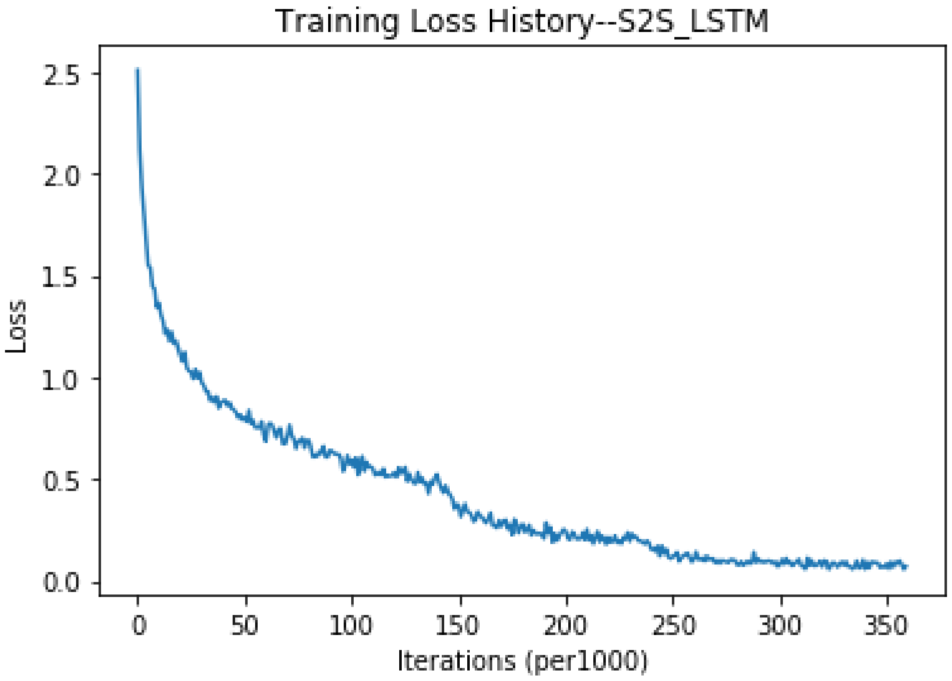
```
def evaluate(test_pairs, encoder, decoder, max_seq_len=max_seq_len)
    encoder.eval()
    decoder.eval()
    with torch.no_grad():
        decoder_words = []
        test_BLEU = []
        for ti in range(len(y_test)):
            test_pair = test_pairs[ti]
            input_tensor = test_pair[0].to(device)
            target_word = y_test[ti]
            #----------sequence to sequence part for encoder----------#
            encoder_hn = encoder.initHidden()
            encoder_cn = encoder.initHidden()
            input_length = input_tensor.size(0)
            for ei in range(input_length):
                _, (encoder_hn, encoder_cn) = encoder(input_tensor[ei], encoder_hn, encoder_cn)
            #----------sequence to sequence part for decoder----------#
            decoder_input = torch.tensor([[SOS_token]], device=device)  # SOS
            decoder_hn = encoder_hn
            decoder_cn = encoder_cn
            decoded_letters = []
            for di in range(max_seq_len):
                decoder_output, (decoder_hn,decoder_cn)= decoder(
                                decoder_input, decoder_hn,decoder_cn)
                topv, topi = decoder_output.data.topk(1)
                if topi.item() == EOS_token:
                    break
                else:
                    decoded_letters.append(characters[topi.item()])
                decoder_input = topi.squeeze().detach()
            ##### token to word #####
            decoder_word = ''.join(decoded_letters)
            decoder_words.append(decoder_word)
            ##### BLEU-4 Score Calculation #####
            word_BLEU = compute_bleu(decoder_word, target_word)
            test_BLEU.append(word_BLEU)
        avg_test_BLEU = np.average(test_BLEU)
```

Evaluation的code如左圖，跟train一樣有encoder跟decoder的串接，然後decoder output出<eos>之後就停止輸出。接著將decoder輸出的token sequence轉成word，最後算BLEU-4分數。

而因為test data有50筆(錯別字,正確字) pairs，所以最後將50個BLEU-4分數取平均。

Results and Discussion

Results and Discussion– Training Loss Curve



左圖是這次所有實驗中最佳訓練的training loss curve，每1000次iterations取值1次，中間有幾階loss明顯下降的部分，看起來是learning_rate階段調降導致的現象。

在整個訓練的過程中，train一個word input就是一次iteration，然後x_train總共有12925個。而我還是有epoch的設計，也就是12925個word train完算一個epoch，並且每個epoch的input word順序都打亂不一致。

較特別的是，我讓teacher forcing ratio隨著epoch數增加而線性調降(即右截圖的Tforce)，idea是嘗試用「師父領進門，修行在個人」的概念來做regularization。

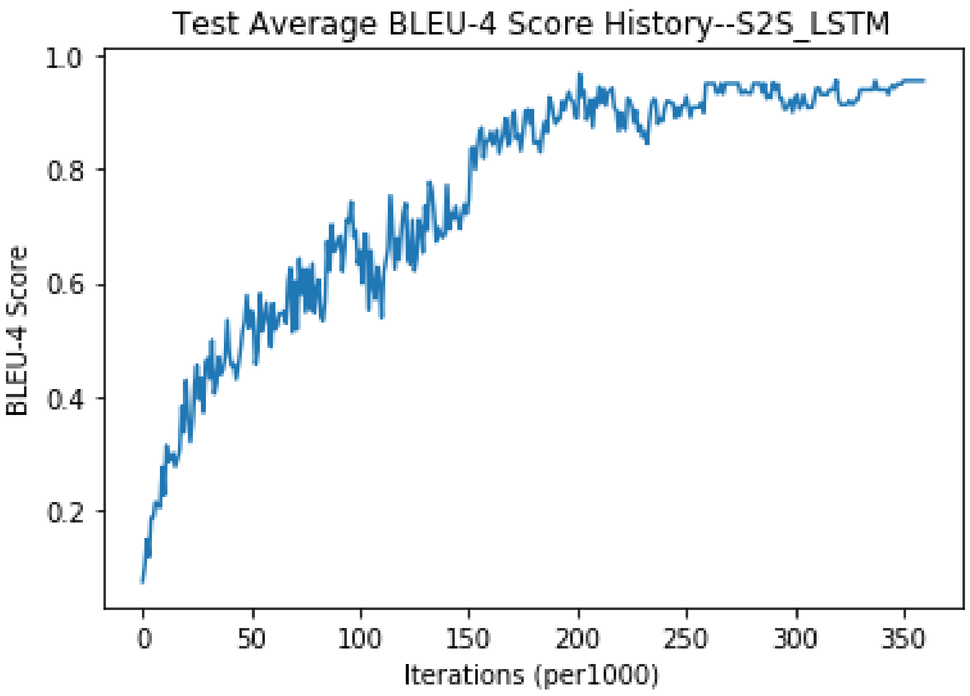
```
################## Start Training Process ######
model_name = 'S2S_LSTM'
hidden_size = 256
epochs = 30
print_every=1000
plot_every=1000
num_iters = len(x_train)  #12925
# num_iters = 1000
Tforce = np.linspace(0.86, 0.6, epochs+1)  #tea

encoder1 = EncoderRNN(vocab_size, hidden_size)
decoder1 = DecoderRNN(hidden_size, vocab_size)
# attn_decoder1 = AttnDecoderRNN(hidden_size, v

start_epo = 1
plot_train_loss = []
plot_test_BLEU = []
for ep in range(start_epo, epochs+1):
    if ep <= 12:
        lr = 0.05
    elif ep <= 20:
        lr = 0.03
    elif ep <= 26:
        lr = 0.01
    else:
        lr = 0.005
```

- Optimizer: SGD
- Loss function: CrossEntropyLoss()

Results and Discussion– BLEU-4 Testing Curve



左圖是這次所有實驗中最佳訓練的BLEU-4 testing curve，每1000次iterations取值1次。而最終最佳的BLEU-4 score為0.9734

其實在epoch=17的某次取值，BLEU就經0.967了，但我還是繼續再Training下去看看，發現後來隨著learning rate & teacher forcing ration都變小，BLEU的variation也跟著變小，而趨於穩定0.95左右(如右圖所示)。

右圖為epoch=30時的每1000次iteration取值截圖。

```
Epoch=30, Learn_rate=0.005, Tforce=0.6000
0m 33s (- 6m 43s) (1000 7%) 0.0713
BLEU-4 score = 0.9491
1m 6s (- 6m 3s) (2000 15%) 0.0684
BLEU-4 score = 0.9491
1m 39s (- 5m 29s) (3000 23%) 0.0877
BLEU-4 score = 0.9547
2m 13s (- 4m 57s) (4000 30%) 0.0623
BLEU-4 score = 0.9547
2m 45s (- 4m 22s) (5000 38%) 0.0953
BLEU-4 score = 0.9554
3m 19s (- 3m 50s) (6000 46%) 0.0702
BLEU-4 score = 0.9554
3m 52s (- 3m 17s) (7000 54%) 0.0939
BLEU-4 score = 0.9554
4m 25s (- 2m 43s) (8000 61%) 0.0863
BLEU-4 score = 0.9554
4m 58s (- 2m 10s) (9000 69%) 0.1011
BLEU-4 score = 0.9554
5m 31s (- 1m 37s) (10000 77%) 0.0820
BLEU-4 score = 0.9547
6m 5s (- 1m 3s) (11000 85%) 0.0587
BLEU-4 score = 0.9554
6m 38s (- 0m 30s) (12000 92%) 0.0735
BLEU-4 score = 0.9547
avg_test_BLEU:0.973
```

計算BLEU-4 score的function未更新前為0.9734
更新後在inference code的結果變為0.9822

Results and Discussion– Results of spelling correction

```
==============================
input:  contenpted
target: contented
pred:   contented
==============================
input:  begining
target: beginning
pred:   beginning
==============================
input:  problam
target: problem
pred:   problem
==============================
input:  dirven
target: driven
pred:   driven
==============================
input:  ecstacy
target: ecstasy
pred:   ecstasy
==============================
input:  juce
target: juice
pred:   juice
==============================
input:  localy
target: locally
pred:   locally
==============================
```

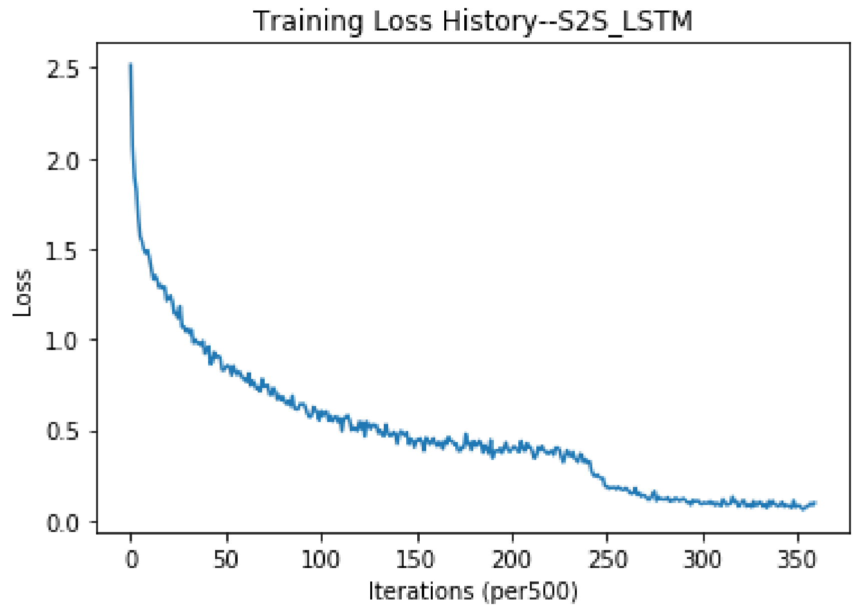
```
==============================
input:  miniscule
target: minuscule
pred:   minuscule
==============================
input:  independant
target: independent
pred:   independent
==============================
input:  aranged
target: arranged
pred:   arranged
==============================
input:  poartry
target: poetry
pred:   portery
==============================
input:  leval
target: level
pred:   level
==============================
input:  basicaly
target: basically
pred:   basically
==============================
input:  triangulaur
target: triangular
pred:   triangular
==============================
```

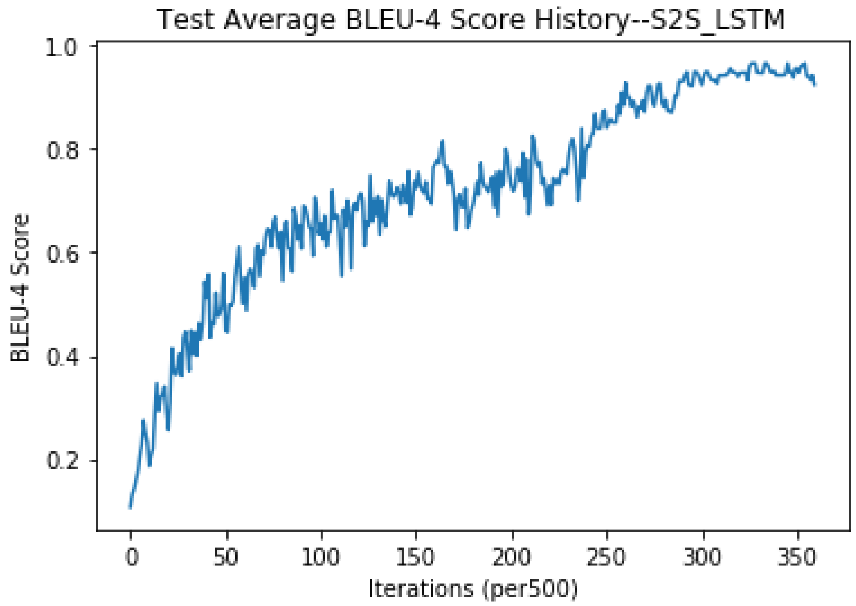
```
==============================
input:  leason
target: lesson
pred:   lesson
==============================
input:  mantain
target: maintain
pred:   maintain
==============================
input:  miricle
target: miracle
pred:   miracle
==============================
input:  oportunity
target: opportunity
pred:   opportunity
==============================
input:  parenthasis
target: parenthesis
pred:   parenthesis
==============================
input:  recetion
target: recession
pred:   recession
==============================
input:  scadual
target: schedule
pred:   schedule
==============================
BLEU-4 score:0.9822
```

實驗結果：左邊為錯別字更正的結果截圖，有頭7字、後7字、以及中間唯一個沒正確更正的" poetry" 。

最右圖的下方有print出此seq2seq模型對於此test data的最佳的BLEU-4 score為0.9822

Results and Discussion– Previous Training





此slide是前面剛training成功的baseline，大部分參數都差不多，唯lr只分兩階段(右上圖)，然後Teacher forcing ratio都是0.8。

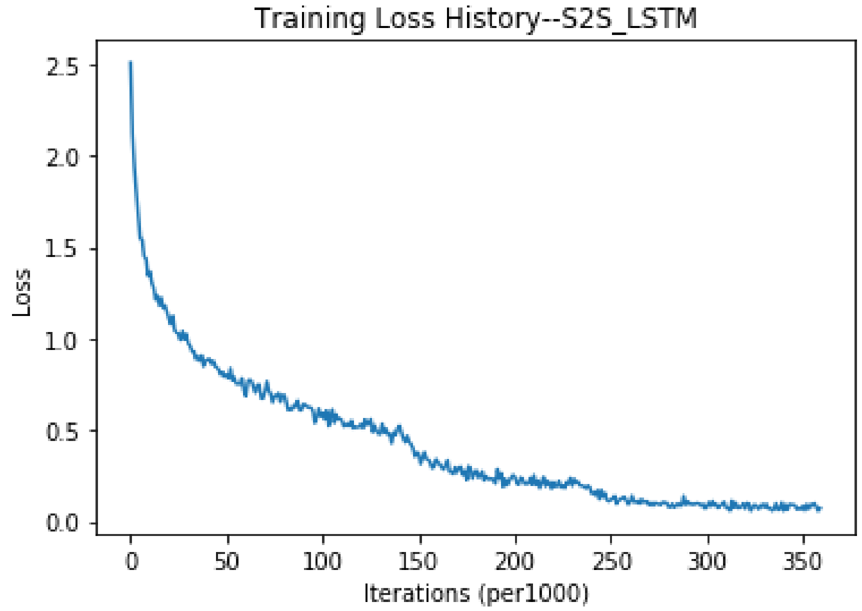
其實這樣的setting，test data的BLEU也達0.9653了，但因為怕說固定的teacher forcing ratio會不會把model train成乖乖牌，所以最後才會有嘗試teacher forcing隨epoch調降的想法。

另外，其實我最一開始是直接用attention based的seq2seq model，但參數不好調，而且traing要很久BLEU也只能到0.86。所以結論是此應用場景看起來不需要attention就可以有不錯的成果。

```
for ep in range(start_epo, epochs+1):
    if ep <= 20:
        lr = 0.05
    else:
        lr = 0.01#0.00001
    tf_ratio = Tforce[ep]
```

```
Epoch=30, Learn_rate=0.01, Tforce=0.8000
0m 32s (- 6m 28s) (1000 7%) 0.1096
BLEU-4 score = 0.9383
Save the best model weights at [Tforce=0.8, Epoch=30, Iter=1000]
1m 4s (- 5m 51s) (2000 15%) 0.0740
BLEU-4 score = 0.9554
Save the best model weights at [Tforce=0.8, Epoch=30, Iter=2000]
1m 35s (- 5m 16s) (3000 23%) 0.0798
BLEU-4 score = 0.9554
Save the best model weights at [Tforce=0.8, Epoch=30, Iter=3000]
2m 6s (- 4m 43s) (4000 30%) 0.0833
BLEU-4 score = 0.9472
2m 38s (- 4m 10s) (5000 38%) 0.0707
BLEU-4 score = 0.9597
Save the best model weights at [Tforce=0.8, Epoch=30, Iter=5000]
3m 9s (- 3m 38s) (6000 46%) 0.0610
BLEU-4 score = 0.9597
Save the best model weights at [Tforce=0.8, Epoch=30, Iter=6000]
3m 40s (- 3m 6s) (7000 54%) 0.0709
BLEU-4 score = 0.9653
Save the best model weights at [Tforce=0.8, Epoch=30, Iter=7000]
4m 12s (- 2m 35s) (8000 61%) 0.0759
BLEU-4 score = 0.9406
4m 43s (- 2m 3s) (9000 69%) 0.0843
BLEU-4 score = 0.9406
5m 15s (- 1m 32s) (10000 77%) 0.0958
BLEU-4 score = 0.9341
5m 46s (- 1m 0s) (11000 85%) 0.0839
BLEU-4 score = 0.9422
6m 17s (- 0m 29s) (12000 92%) 0.0968
BLEU-4 score = 0.9241
avg_test_BLEU:0.940
```

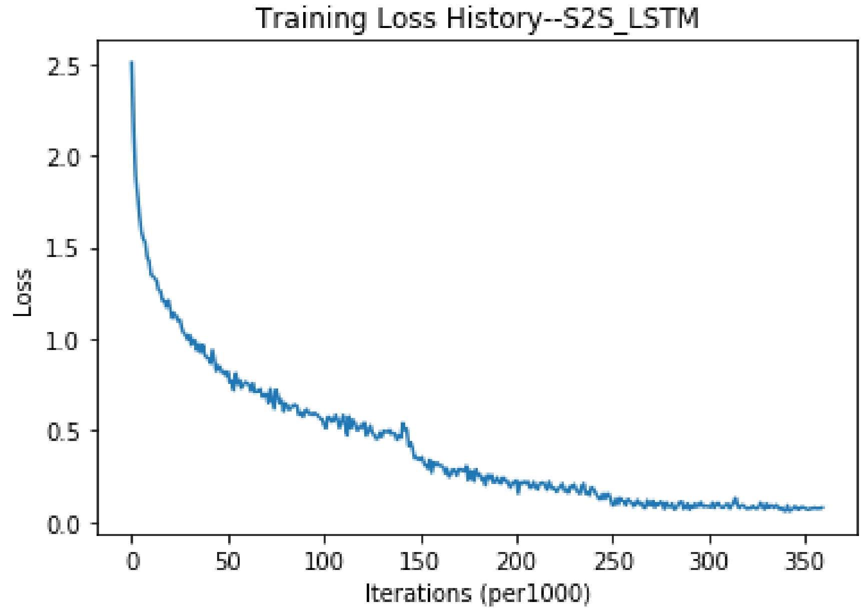
助教BLEU function修改後的補充training

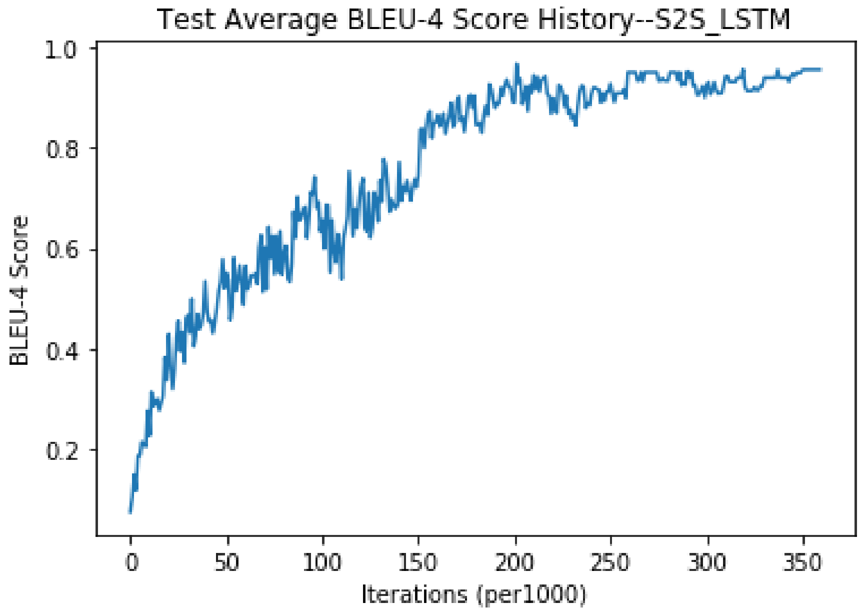


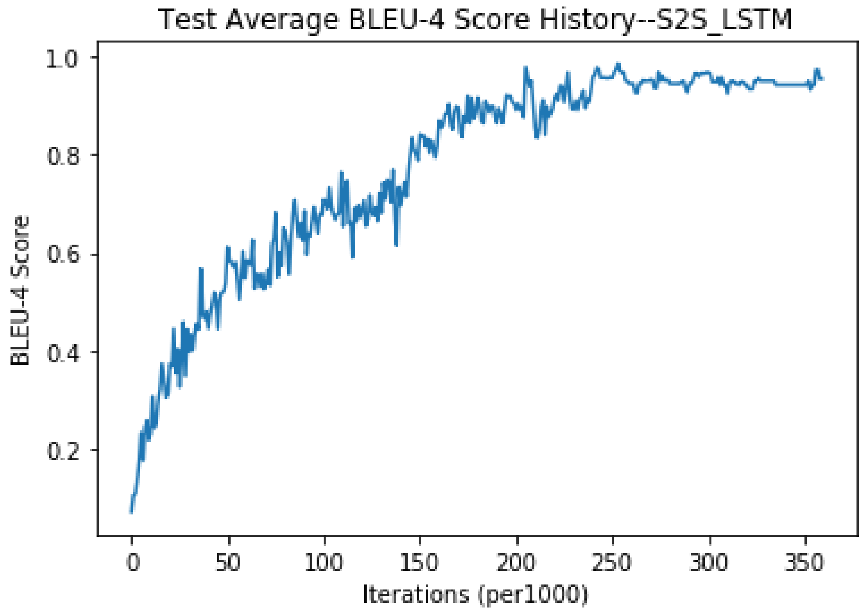
左圖為修改前
右圖為修改後

大致上趨勢與階段變化都差不多，因為修改的是BLEU function的參數，不是model的參數。

而修改後的BLEU score整體上有上升快一些。







```
Epoch= 1, Learn_rate=0.05, Tforce=0.8510
0m 32s (- 6m 24s) (1000 7%) 2.5076
BLEU-4 score = 0.0725
1m 3s (- 5m 47s) (2000 15%) 2.1341
BLEU-4 score = 0.1072
1m 35s (- 5m 16s) (3000 23%) 1.8877
BLEU-4 score = 0.1096
2m 7s (- 4m 44s) (4000 30%) 1.7739
BLEU-4 score = 0.1402
2m 39s (- 4m 12s) (5000 38%) 1.6736
BLEU-4 score = 0.1771
3m 10s (- 3m 40s) (6000 46%) 1.5838
BLEU-4 score = 0.2338
3m 41s (- 3m 7s) (7000 54%) 1.5442
BLEU-4 score = 0.1777
4m 12s (- 2m 35s) (8000 61%) 1.5252
BLEU-4 score = 0.2437
4m 44s (- 2m 3s) (9000 69%) 1.4442
BLEU-4 score = 0.2592
5m 15s (- 1m 32s) (10000 77%) 1.4258
BLEU-4 score = 0.2194
5m 46s (- 1m 0s) (11000 85%) 1.3499
BLEU-4 score = 0.2333
6m 17s (- 0m 29s) (12000 92%) 1.3428
BLEU-4 score = 0.3065
avg_test_BLEU:0.282
```

```
Epoch=22, Learn_rate=0.01, Tforce=0.6690
0m 32s (- 6m 27s) (1000 7%) 0.1277
BLEU-4 score = 0.9738
1m 3s (- 5m 46s) (2000 15%) 0.0956
BLEU-4 score = 0.9837
Save the best model weights at [Tforce=0.669,
Epoch=22, Iter=2000]
1m 35s (- 5m 14s) (3000 23%) 0.1296
BLEU-4 score = 0.9692
2m 6s (- 4m 42s) (4000 30%) 0.1165
BLEU-4 score = 0.9658
2m 38s (- 4m 11s) (5000 38%) 0.1124
BLEU-4 score = 0.9658
3m 10s (- 3m 39s) (6000 46%) 0.0904
BLEU-4 score = 0.9487
3m 41s (- 3m 7s) (7000 54%) 0.1238
BLEU-4 score = 0.9483
4m 12s (- 2m 35s) (8000 61%) 0.1129
BLEU-4 score = 0.9426
4m 44s (- 2m 4s) (9000 69%) 0.1063
BLEU-4 score = 0.9426
5m 15s (- 1m 32s) (10000 77%) 0.0855
BLEU-4 score = 0.9431
5m 47s (- 1m 0s) (11000 85%) 0.0942
BLEU-4 score = 0.9250
6m 18s (- 0m 29s) (12000 92%) 0.1188
BLEU-4 score = 0.9244
avg_test_BLEU:0.925
```

```
Epoch=30, Learn_rate=0.005, Tforce=0.6000
0m 32s (- 6m 26s) (1000 7%) 0.0792
BLEU-4 score = 0.9414
1m 4s (- 5m 50s) (2000 15%) 0.0800
BLEU-4 score = 0.9414
1m 35s (- 5m 16s) (3000 23%) 0.0746
BLEU-4 score = 0.9414
2m 7s (- 4m 44s) (4000 30%) 0.0690
BLEU-4 score = 0.9414
2m 39s (- 4m 12s) (5000 38%) 0.0666
BLEU-4 score = 0.9494
3m 10s (- 3m 39s) (6000 46%) 0.0725
BLEU-4 score = 0.9308
3m 41s (- 3m 7s) (7000 54%) 0.0728
BLEU-4 score = 0.9414
4m 12s (- 2m 35s) (8000 61%) 0.0771
BLEU-4 score = 0.9414
4m 44s (- 2m 4s) (9000 69%) 0.0715
BLEU-4 score = 0.9717
5m 16s (- 1m 32s) (10000 77%) 0.0749
BLEU-4 score = 0.9717
5m 47s (- 1m 0s) (11000 85%) 0.0723
BLEU-4 score = 0.9530
6m 19s (- 0m 29s) (12000 92%) 0.0772
BLEU-4 score = 0.9530
avg_test_BLEU:0.953
```

上三個圖分別為最後一次train model的epoch=1, 22, 30之每1000次iteration的training取值截圖，而這最後一次model training之最大BLEU-4 score出現在epoch=22的第2000次iteration。此時的learning_rate=0.01, teacher_forcing_ratio=0.669。

```
==========================
input:  contenpted
target: contented
pred:   conpetented
==========================
input:  begining
target: beginning
pred:   beginning
==========================
input:  problam
target: problem
pred:   problem
==========================
input:  dirven
target: driven
pred:   driven
==========================
input:  ecstacy
target: ecstasy
pred:   ecstasy
==========================
input:  juce
target: juice
pred:   juice
==========================
input:  localy
target: locally
pred:   locally
==========================
```

```
==========================
input:  enxt
target: next
pred:   next
==========================
input:  powerfull
target: powerful
pred:   powerful
==========================
input:  practial
target: practical
pred:   practical
==========================
input:  repatition
target: repartition
pred:   repetition
==========================
input:  repentence
target: repentance
pred:   repentance
==========================
input:  substracts
target: subtracts
pred:   subtracts
==========================
input:  beed
target: bead
pred:   bead
==========================
```

```
==========================
input:  leason
target: lesson
pred:   lesson
==========================
input:  mantain
target: maintain
pred:   maintain
==========================
input:  miricle
target: miracle
pred:   miracle
==========================
input:  oportunity
target: opportunity
pred:   opportunity
==========================
input:  parenthasis
target: parenthesis
pred:   parenthesis
==========================
input:  recetion
target: recession
pred:   recession
==========================
input:  scadual
target: schedule
pred:   schedule
==========================
BLEU-4 score:0.9837
```

最後一次實驗結果：左邊為錯別字更正的結果截圖，有頭7字、後7字、以及中間唯一個沒正確更正的”repatition”，不過在這裡仔細想想，model會改正在”repetition”，好像也不是沒道理。

最右圖的下方有print出此seq2seq模型對於此test data的最佳的BLEU-4 score為0.9837

The End