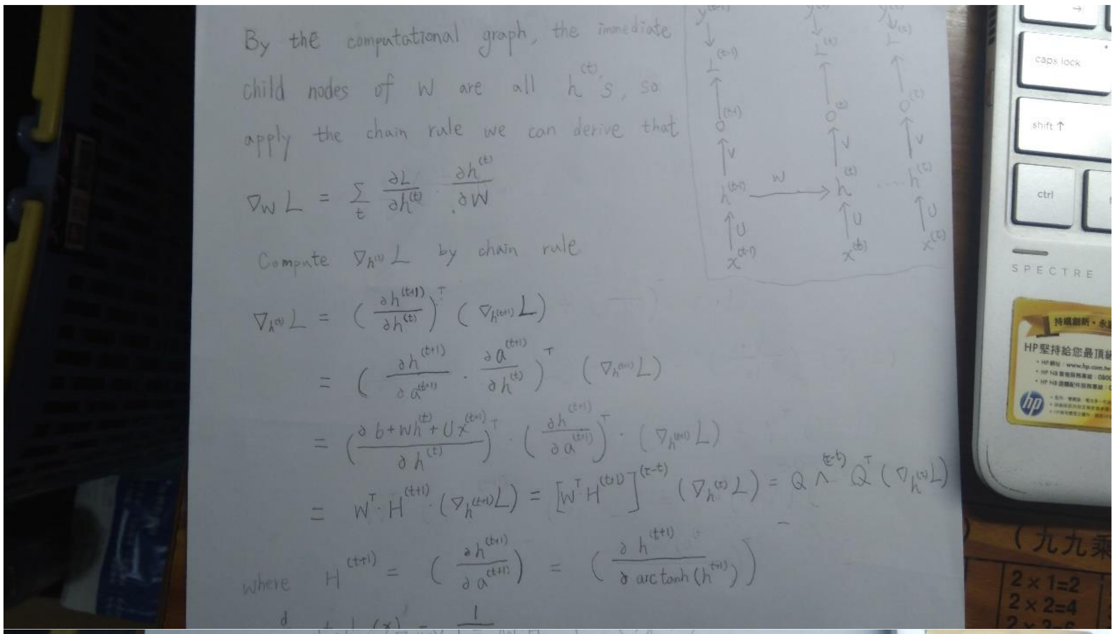
Lab4

0856018 謝宗祐

1. Introduction

   In this lab, I have to implement a seq2seq model, which can do the spell correction to the word. In addition, the encoder and the decoder of the seq2seq model are both LSTM.

2. Derivation of BPTT



By the computational graph, the immediate child nodes of W are all $h^{(t)}$'s, so apply the chain rule we can derive that

$$\nabla_W L = \sum_t \frac{\partial L}{\partial h^{(t)}} \frac{\partial h^{(t)}}{\partial W}$$

Compute $\nabla_{h^{(t)}} L$ by chain rule

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

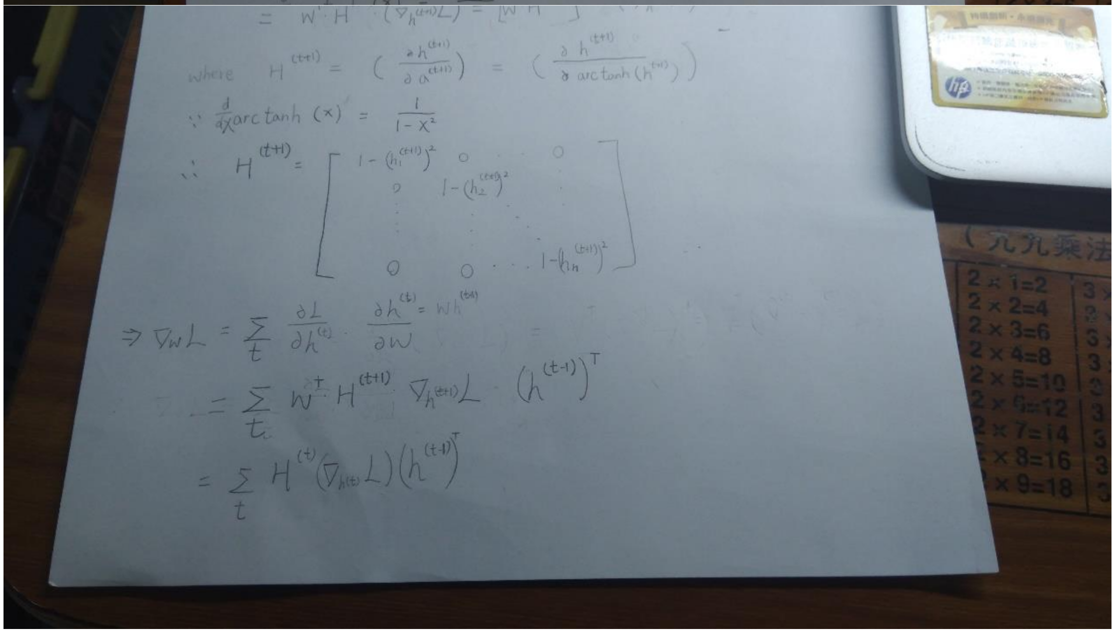
$$= \left(\frac{\partial h^{(t+1)}}{\partial a^{(t+1)}} \cdot \frac{\partial a^{(t+1)}}{\partial h^{(t)}}\right)^T (\nabla_{h^{(t+1)}} L)$$

$$= \left(\frac{\partial\, b + Wh^{(t)} + Ux^{(t+1)}}{\partial h^{(t)}}\right)^T \cdot \left(\frac{\partial h^{(t+1)}}{\partial a^{(t+1)}}\right)^T \cdot (\nabla_{h^{(t+1)}} L)$$

$$= W^T \cdot H^{(t+1)} \cdot (\nabla_{h^{(t+1)}} L) = [W^T H^{(t+1)}]^{(\tau - t)} (\nabla_{h^{(\tau)}} L) = Q \Lambda^{(\tau-t)} Q^T (\nabla_{h^{(\tau)}} L)$$

where $H^{(t+1)} = \left(\frac{\partial h^{(t+1)}}{\partial a^{(t+1)}}\right) = \left(\frac{\partial h^{(t+1)}}{\partial \operatorname{arctanh}(h^{(t+1)})}\right)$

$y^{(t-1)}$
$L^{(t-1)}$
$o^{(t-1)}$
V
$h^{(t-1)}$
U
$x^{(t-1)}$
W
$y^{(t)}$
$L^{(t)}$
$o^{(t)}$
V
$h^{(t)}$
U
$x^{(t)}$
$y^{(\tau)}$
$L^{(\tau)}$
$o^{(\tau)}$
V
$h^{(\tau)}$
U
$x^{(\tau)}$



where $H^{(t+1)} = \left(\frac{\partial h^{(t+1)}}{\partial a^{(t+1)}}\right) = \left(\frac{\partial h^{(t+1)}}{\partial \operatorname{arctanh}(h^{(t+1)})}\right)$

$\because \frac{d}{dx}\operatorname{arctanh}(x) = \frac{1}{1-x^2}$

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

$$\Rightarrow \nabla_W L = \sum_t \frac{\partial L}{\partial h^{(t)}} \cdot \frac{\partial h^{(t)}}{\partial W} \quad \frac{\partial h^{(t)}}{\partial W} = Wh^{(t-1)}$$

$$= \sum_t W^T \cdot H^{(t+1)} \nabla_{h^{(t+1)}} L \; (h^{(t-1)})^T$$

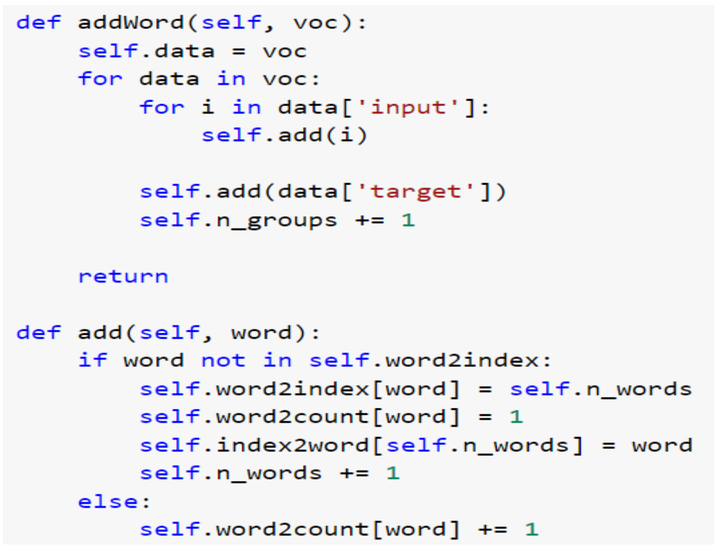
$$= \sum_t H^{(t)} (\nabla_{h^{(t)}} L)(h^{(t-1)})^T$$

3. Implementation details

   (1). Describe how you implement your model

   I follow the reference 1 given by TA and sample.py to build my model

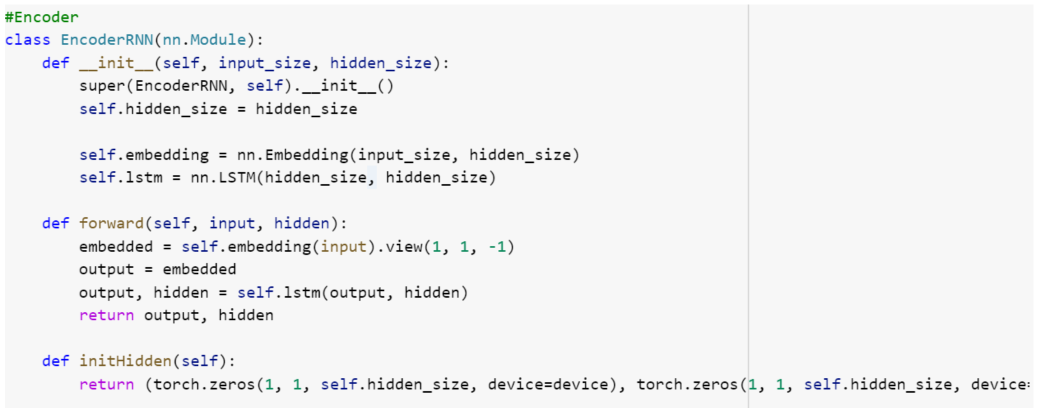
   A. Dataloader



```python
def addWord(self, voc):
    self.data = voc
    for data in voc:
        for i in data['input']:
            self.add(i)

        self.add(data['target'])
        self.n_groups += 1

    return

def add(self, word):
    if word not in self.word2index:
        self.word2index[word] = self.n_words
        self.word2count[word] = 1
        self.index2word[self.n_words] = word
        self.n_words += 1
    else:
        self.word2count[word] += 1
```

My dataloader can count the number of words we have added, so for a new coming word, I set the index of this word is how many words I have added. Word2index is a dictionary which can map each word to a specific index, index2word is another dictionary which maps the index to the word. In addition, both input and target need to append a EOS token.

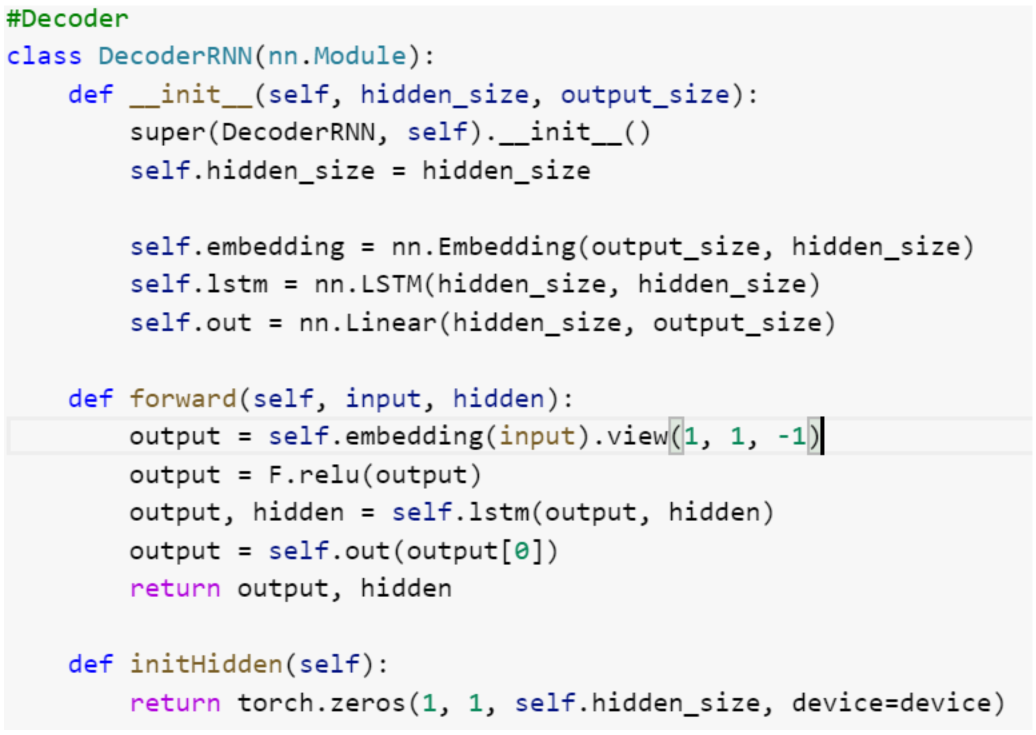
B. Encoder



```python
#Encoder
class EncoderRNN(nn.Module):
    def __init__(self, input_size, hidden_size):
        super(EncoderRNN, self).__init__()
        self.hidden_size = hidden_size

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

    def forward(self, input, hidden):
        embedded = self.embedding(input).view(1, 1, -1)
        output = embedded
        output, hidden = self.lstm(output, hidden)
        return output, hidden

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

The input of the encoder is the index of the word, first embed it to the vector and then feed to the LSTM. Note that the hidden of LSTM is a tuple which contains hidden state and cell state.

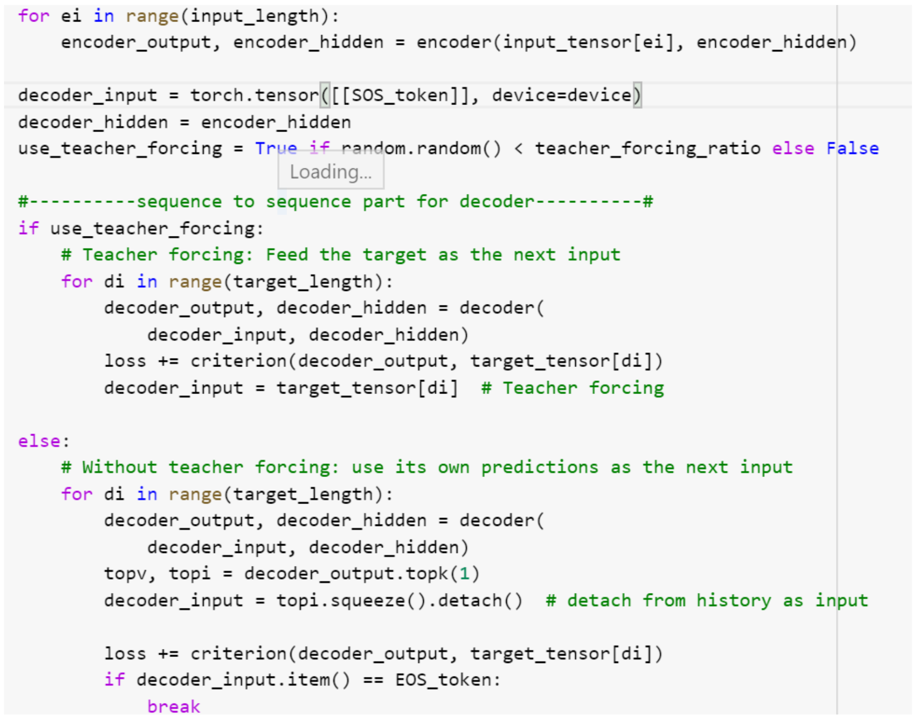
C. Decoder



```
#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.lstm = nn.LSTM(hidden_size, hidden_size)
        self.out = nn.Linear(hidden_size, output_size)

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

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

Like the decoder in sample.py, the only difference is that I replace the GRU to LSTM, and remove the LogSoftMax layer since I use the crossentropyloss.

D. Training function



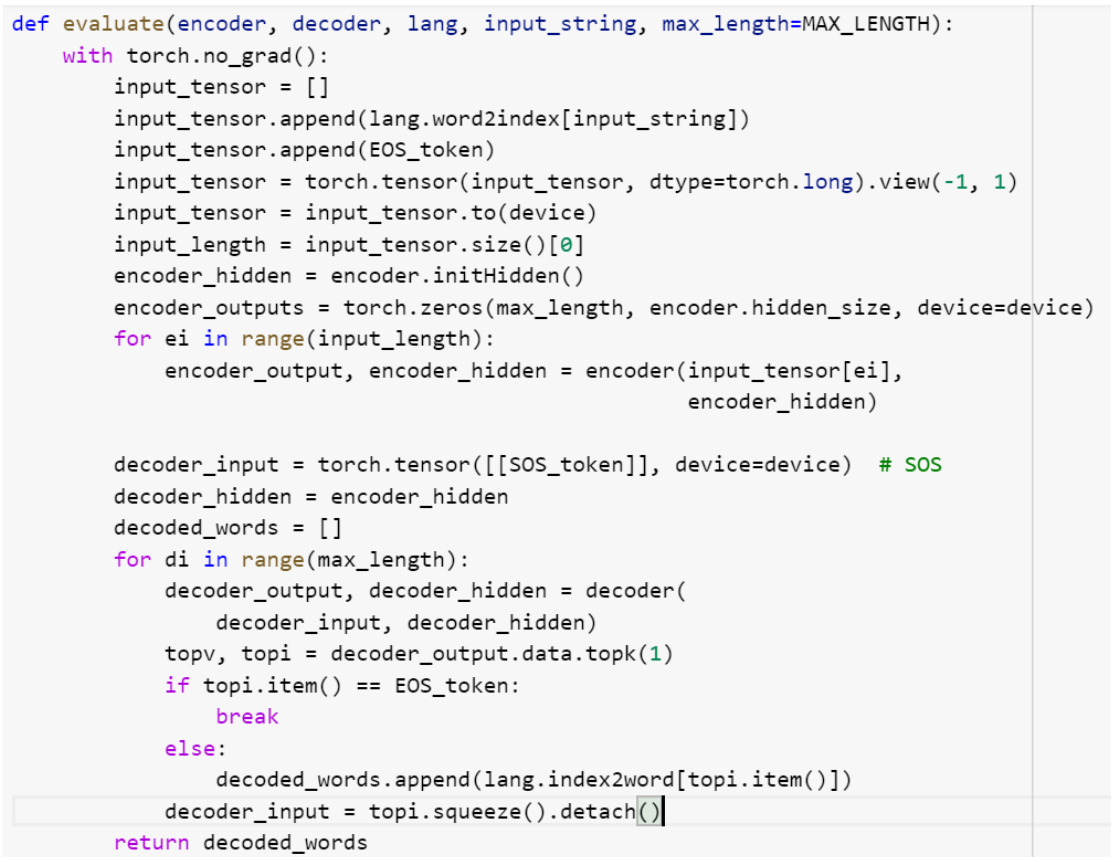
```
for ei in range(input_length):
    encoder_output, encoder_hidden = encoder(input_tensor[ei], encoder_hidden)

decoder_input = torch.tensor([[SOS_token]], device=device)
decoder_hidden = encoder_hidden
use_teacher_forcing = True if random.random() < teacher_forcing_ratio else False

#----------sequence to sequence part for decoder----------#
if use_teacher_forcing:
    # Teacher forcing: Feed the target as the next input
    for di in range(target_length):
        decoder_output, decoder_hidden = decoder(
            decoder_input, decoder_hidden)
        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(
            decoder_input, decoder_hidden)
        topv, topi = decoder_output.topk(1)
        decoder_input = topi.squeeze().detach()  # detach from history as input

        loss += criterion(decoder_output, target_tensor[di])
        if decoder_input.item() == EOS_token:
            break
```

Like the training function in sample.py, but I have to revise the encoder part even though there is only an input word. Because I add

EOS token to the end of every input, my input length is at least 2.

There are totally 100 epochs for training, and each epoch train the model for whole data in train.json.

(2). Screenshot of the evaluation part

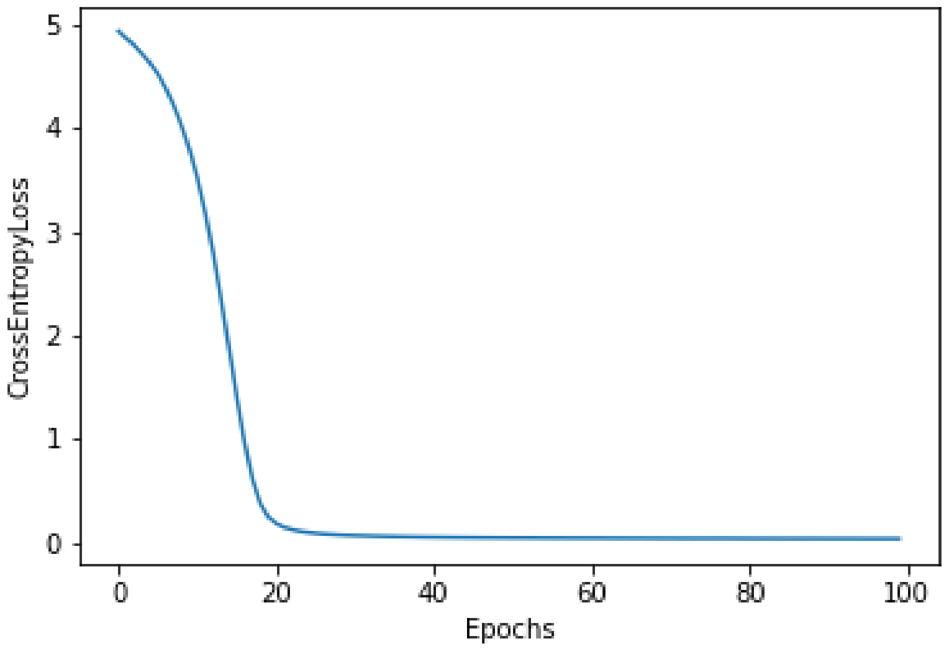


```python
def evaluate(encoder, decoder, lang, input_string, max_length=MAX_LENGTH):
    with torch.no_grad():
        input_tensor = []
        input_tensor.append(lang.word2index[input_string])
        input_tensor.append(EOS_token)
        input_tensor = torch.tensor(input_tensor, dtype=torch.long).view(-1, 1)
        input_tensor = input_tensor.to(device)
        input_length = input_tensor.size()[0]
        encoder_hidden = encoder.initHidden()
        encoder_outputs = torch.zeros(max_length, encoder.hidden_size, device=device)
        for ei in range(input_length):
            encoder_output, encoder_hidden = encoder(input_tensor[ei],
                                                     encoder_hidden)

        decoder_input = torch.tensor([[SOS_token]], device=device)  # SOS
        decoder_hidden = encoder_hidden
        decoded_words = []
        for di in range(max_length):
            decoder_output, decoder_hidden = decoder(
                decoder_input, decoder_hidden)
            topv, topi = decoder_output.data.topk(1)
            if topi.item() == EOS_token:
                break
            else:
                decoded_words.append(lang.index2word[topi.item()])
            decoder_input = topi.squeeze().detach()
        return decoded_words
```

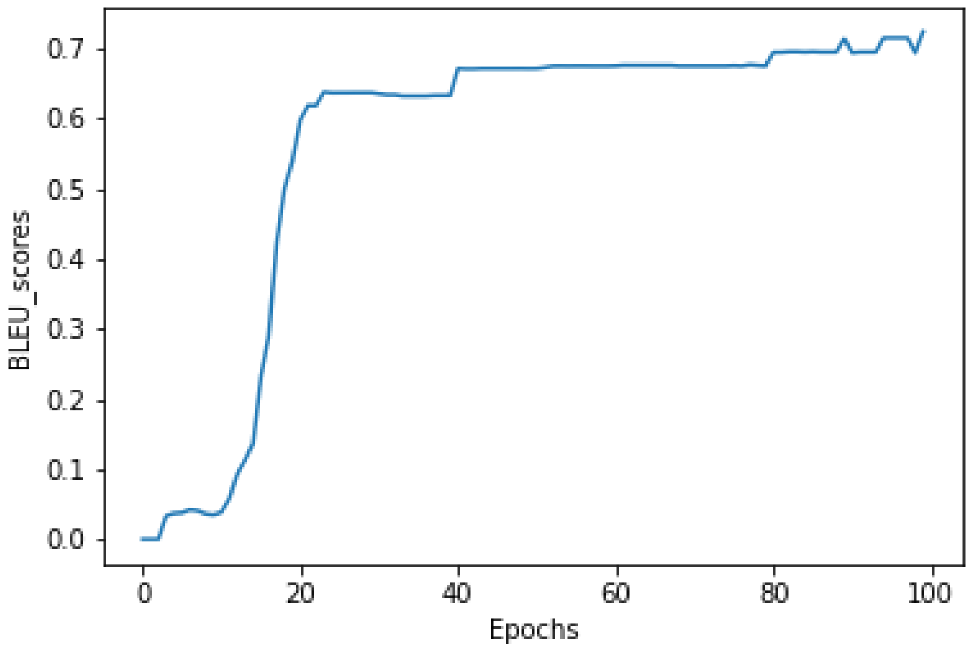
This is my evaluation function, and the parameters of this function doesn't have the target string, only use input_string and the model to predict the output.

4. Results and discussion

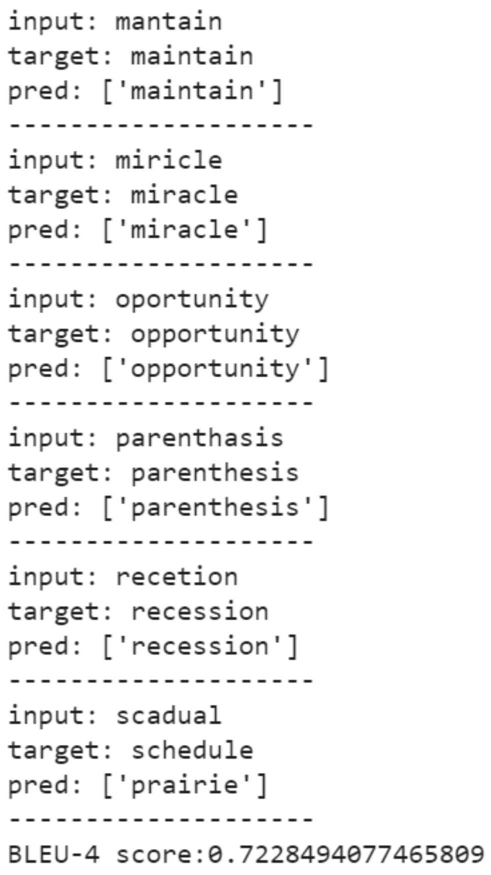
(1). Training loss curve and BLEU-4 score testing curve



CrossEntropyLoss
Epochs



BLEU_scores
Epochs



```
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: ['prairie']
--------------------
BLEU-4 score:0.7228494077465809
```

I set the hidden size = 256, vocab_size = 17703, teacher forcing ratio = 0.7, learning rate = 0.05, Max_length = 10. Finally, after 100 epochs of training, I get about 0.72 BLEU-4 score on the test data.

(2). Discussion of the results

A. By the result of evaluation test data, first I found that my model can't predict the words that do not appear in the training data. Second, the previous 6 words in the test.json almost predicted wrong by my model, all these words have appeared in train.json, so I think maybe my model forget these words.

B. Because I train so many epochs, the teacher forcing ratio doesn't have any influence on my results, both the loss and BLEU score.