Lab4

> 0856029 Yu Shao, Liu 劉昱劭

Introduction

In this lab, we implement a seq2seq encoder-decoder network with recurrent units for English spelling correction. We use LSTM to build Encode and Decoder.

BPTT

First, calculate $\nabla_W{}^L$

$$\nabla_W{}^L = \sum_t \sum_i (\frac{\partial L}{\partial h_i^{(t)}}) \nabla_W{}^{h_i^{(t)}}, \nabla_W{}^{h_i^{(t)}} = \frac{\partial h_i^{(t)}}{\partial a_i^{(t)}} \frac{\partial a_i^{(t)}}{\partial W}$$

$$\frac{\partial h_i^{(t)}}{\partial a_i^{(t)}} = tanh'(a_i^{(t)}) = 1 - tanh^2(a_i^{(t)}) = 1 - (h_i^{(t)})^2, \frac{\partial a_i^{(t)}}{\partial W} = h_i^{(t-1)}$$

$$\nabla_W{}^{h_i^{(t)}} = (1 - (h_i^{(t)})^2)(h_i^{(t-1)})$$

Second, calculate $\frac{\partial L}{\partial h_i^{(t)}}$.

$$\frac{\partial L}{\partial h_i^{(t)}} = (\frac{\partial h_i^{(t+1)}}{\partial h_i^{(t)}} \frac{\partial L}{\partial h_i^{(t+1)}}) + (\frac{\partial o_i^{(t)}}{\partial h_i^{(t)}} \frac{\partial L}{\partial o_i^{(t)}})$$

$$\frac{\partial h_i^{(t+1)}}{\partial h_i^{(t)}} = \frac{\partial a_i^{(t+1)}}{\partial h_i^{(t)}} \frac{\partial h_i^{(t+1)}}{\partial a_i^{(t+1)}} = W(1 - (h_i^{(t+1)})^2)$$

$$\frac{\partial o_i^{(t)}}{\partial h_i^{(t)}} = V$$

Third, calculate $\frac{\partial L}{\partial o_i^{(t)}}$

$$\frac{\partial L}{\partial o_i^{(t)}} = \frac{\partial \hat{y}^{(t)}}{\partial o_i^{(t)}} \frac{\partial L}{\partial \hat{y}^{(t)}} = \text{softmax}'(o^{(t)})(\frac{\partial - \sum_j y_j^{(t)} log(\hat{y}_j^{(t)})}{\partial \hat{y}^{(t)}})$$

$$textsoftmax'(x) = \begin{cases} \text{softmax}(x_i)(1 - \text{softmax}(x_i)), \text{if i==j} \\ -\text{softmax}(x_i)\text{softmax}(x_j), \text{if i != j} \end{cases}$$

$$= \hat{y}_i^{(t)}(1 - \hat{y}_i^{(t)})(-\frac{y_i^{(t)}}{\hat{y}_i^{(t)}}) + \sum_{j \neq i} -\hat{y}_i^{(t)}\hat{y}_j^{(t)}(-\frac{y_j^{(t)}}{\hat{y}_j^{(t)}})$$

$$= (\hat{y}_i^{(t)} - 1)y_i^{(t)} + \sum_{j \neq i} \hat{y}_i^{(t)} y_j^{(t)} = (\sum_j y_j^{(t)})\hat{y_i}^{(t)} - y_i^{(t)}$$

if $\sum y(t)$ is one, equation can rewrite to:

$$\frac{\partial L}{\partial o_i^{(t)}} = \hat{y_i^{(t)}} - y_i^{(t)}$$

And need final time layer hidden gradient to start BPTT

And need final time layer hidden gradient to start BPTT.

$$\nabla_{h_i^{(\tau)}} L = \frac{\partial o_i^{(\tau)}}{\partial h_i^{(\tau)}} \frac{\partial L}{\partial o_i^{(\tau)}} = V(\hat{y_i^{(\tau)}} - y_i^{(\tau)})$$

Finally, get all equations for computing gradient :

$$\frac{\partial L}{\partial h_i^{(t)}} = W(1 - (h_i^{t+1})^2) \frac{\partial L}{\partial h_i^{(t+1)}} + V(\hat{y_i^{(t)}} - y_i^{(t)})$$

$$\nabla_W L = \sum_t \sum_i (\frac{\partial L}{\partial h_i^{(t)}})(\frac{\partial h_i^{(t)}}{\partial W}) = \sum_t \sum_i \frac{\partial L}{\partial h_i^{(t)}}(1 - (h_i^{(t)})^2)(h_i^{(t-1)})$$

$$\nabla_U L = \sum_t \sum_i (\frac{\partial L}{\partial h_i^{(t)}})(\frac{\partial h_i^{(t)}}{\partial U}) = \sum_t \sum_i \frac{\partial L}{\partial h_i^{(t)}}(1 - (h_i^{(t)})^2)(x_i^{(t)})$$

$$\nabla_V L = \sum_t \sum_i (\frac{\partial L}{\partial o_i^{(t)}})(\frac{\partial o_i^{(t)}}{\partial V}) = \sum_t \sum_i (\hat{y_i^{(t)}} - y_i^{(t)}) h_i^{(t)}$$

$$\nabla_b L = \sum_t \sum_i (\frac{\partial L}{\partial h_i^{(t)}})(\frac{\partial h_i^{(t)}}{\partial b}) = \sum_t \sum_i \frac{\partial L}{\partial h_i^{(t)}}(1 - (h_i^{(t)})^2)$$

$$\nabla_c L = \sum_t \sum_i (\frac{\partial L}{\partial o_i^{(t)}})(\frac{\partial o_i^{(t)}}{\partial c}) = \sum_t \sum_i \hat{y_i^{(t)}} - y_i^{(t)}$$

Explanation

Encoder

In [3]:

```
from __future__ import unicode_literals, print_function, division
import torch
import torch.nn as nn
import torch.nn.functional as F
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

```
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.rnn = nn.LSTM(hidden_size, hidden_size)

    def forward(self, input, hidden_state, cell_state):
        embedded = self.embedding(input).view(1, 1, -1)  # view(1,1,-1)
 due to input of rnn must be (seq_len,batch,vec_dim)
        output,(hidden_state,cell_state) = self.rnn(embedded, (hidden_s
tate,cell_state) )
        return output,hidden_state,cell_state

    def init_h0(self):
        return torch.zeros(1, 1, self.hidden_size, device=device)
    def init_c0(self):
        return torch.zeros(1, 1, self.hidden_size, device=device)
```

Decoder

In [ ]:

```
class DecoderRNN(nn.Module):
    def __init__(self, input_size, hidden_size):
        super(DecoderRNN, self).__init__()
        self.hidden_size = hidden_size
        self.embedding = nn.Embedding(input_size, hidden_size)
        self.rnn = nn.LSTM(hidden_size, hidden_size)
        self.out = nn.Linear(hidden_size, input_size)
        self.softmax = nn.LogSoftmax(dim=1)

    def forward(self, input, hidden_state, cell_state):
        output = self.embedding(input).view(1, 1, -1)
        output = F.relu(output)
        output, (hidden_state,cell_state) = self.rnn(output, (hidden_st
ate,cell_state) )
        output = self.softmax(self.out(output[0]))
        return output,hidden_state,cell_state
```

```
    def init_h0(self):
        """
        :return: (num_layers * num_directions, batch, hidden_size)
        """
        return torch.zeros(1, 1, self.hidden_size, device=device)
    def init_c0(self):
        """
        :return: (num_layers * num_directions, batch, hidden_size)
        """
        return torch.zeros(1, 1, self.hidden_size, device=device)
```

Evaluation using BLEU-4

- load testing data

```
testing_list,testing_input=data.build_training_set(path='test.json')
testing_tensor_list=[]
```

- testing

```
# testing using bleu-4
score=0
for i,(_,target) in enumerate(testing_input):
    if not verbose:
        print(f'input:  {_}')
        print(f'target: {target}')
        print(f'pred:   {predict}')
        print('='*28)
    predict=data.idx2seq(predicted_list[i])
    score+=compute_bleu(predict,target)
score/=len(testing_input)
print(f'BLEU-4: {score:.2f}')

loss_list.append(loss)
BLEU_list.append(score)
```

Results & Discussion

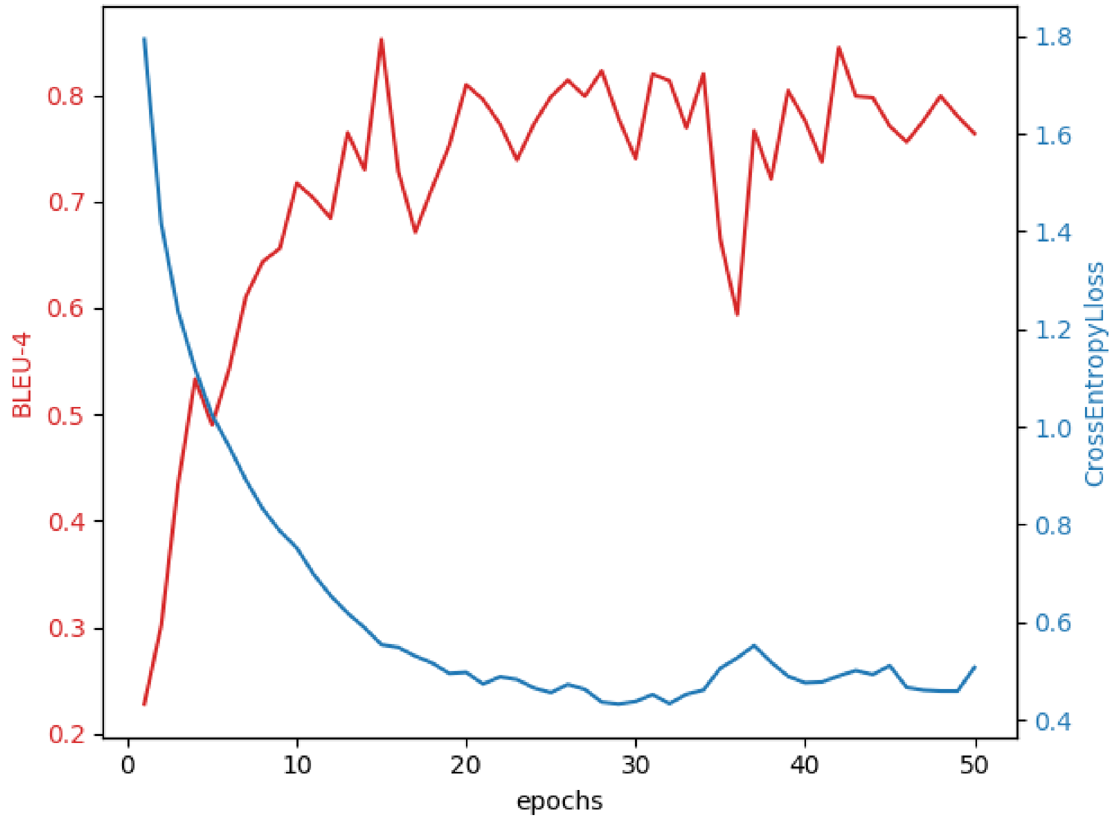
output sample

```
input:  parenthasis
target: parenthesis
pred:   paranishess
=============================
input:  recetion
target: recession
pred:   recoitin
=============================
input:  scadual
target: schedule
pred:   scadual
=============================
BLEU-4: 0.40
```

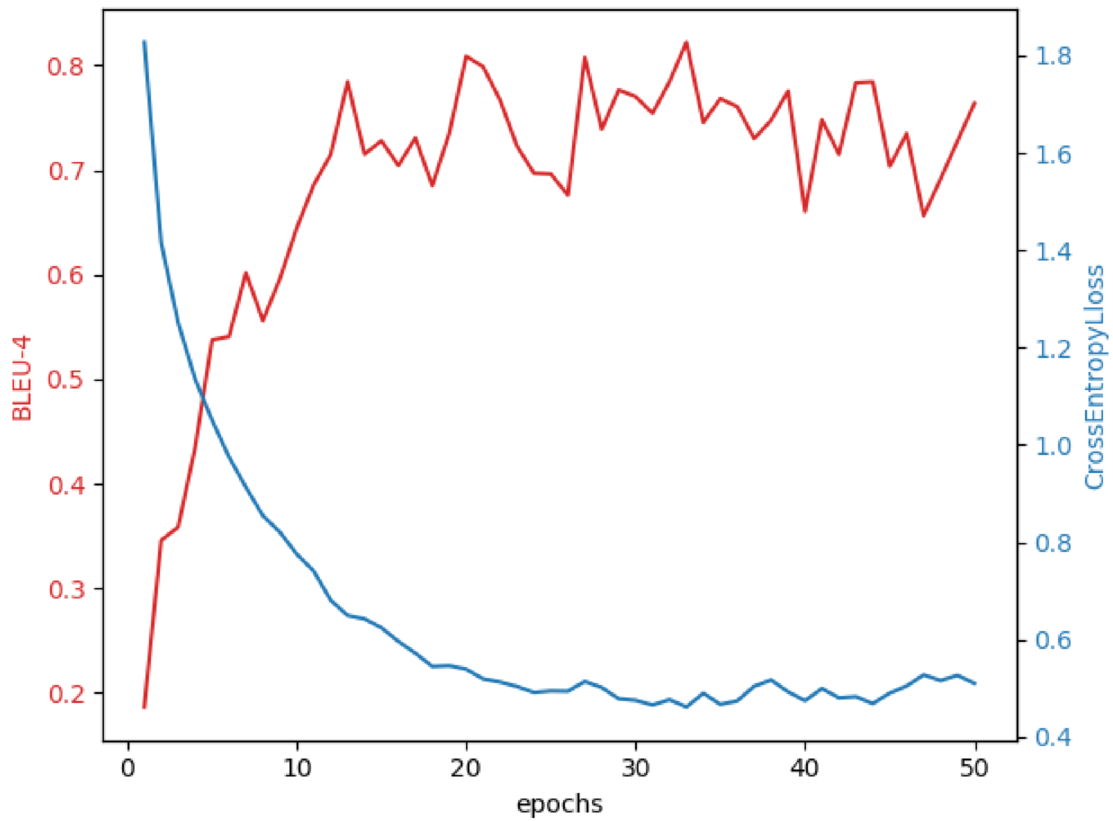
Try 256 hidden units

We could see that large teacher forcing ratio might improve BLEU-4 score faster a little bit.

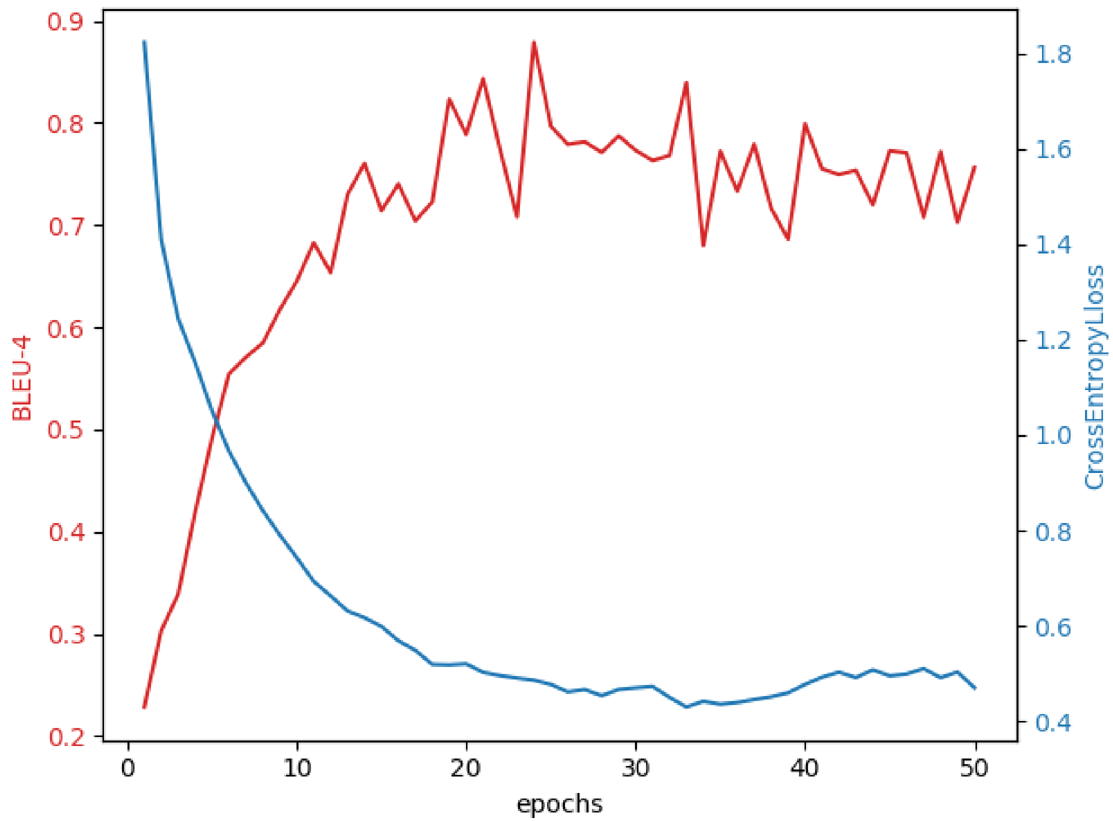
- teacher_forcing ratio = 1



- teacher_forcing ratio = 0.5



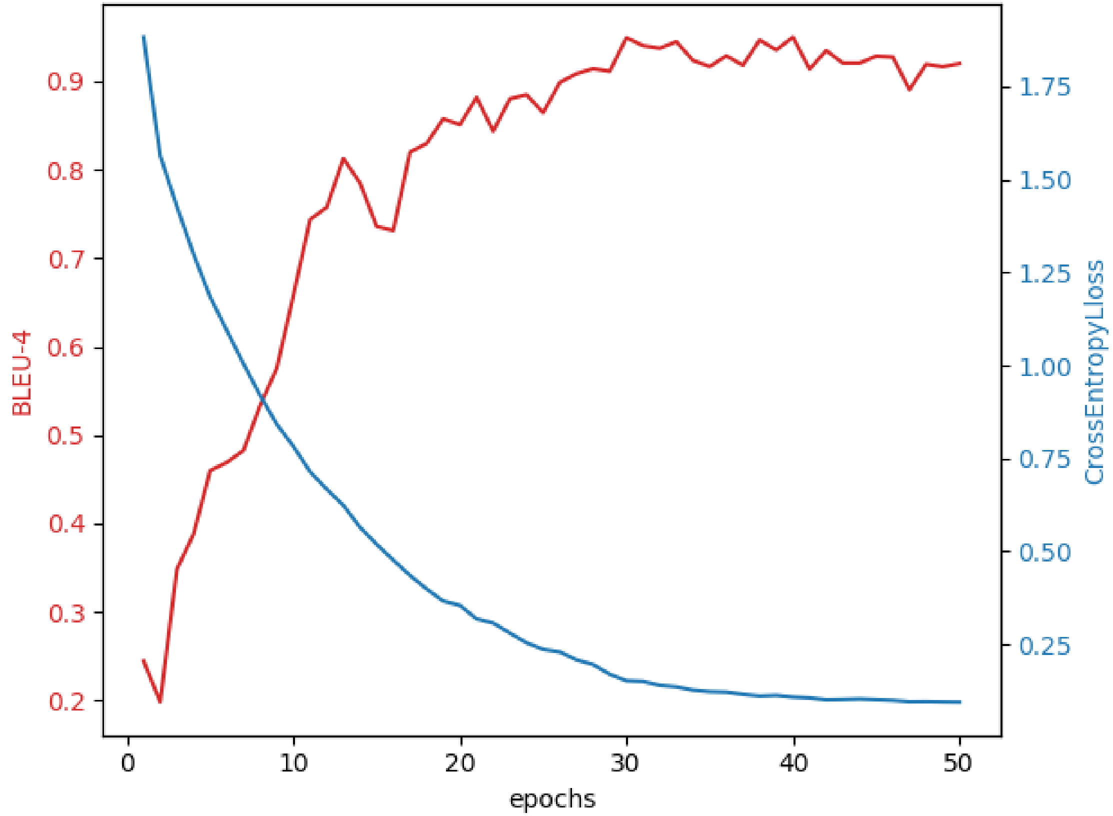
- teacher_forcing ratio = 0



Try 512 hidden units

We could see using 512 hidden units could achieve higher BLEU-4 score (about 0.95) and learning curve is much more smooth

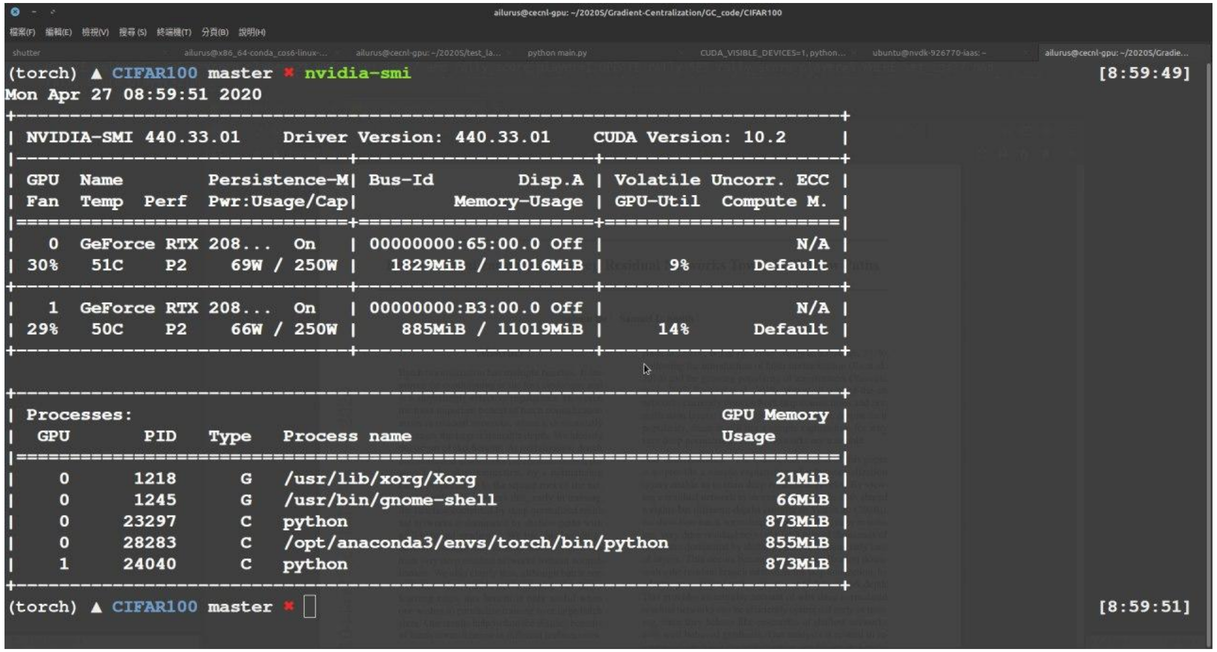
- teacher_forcing ratio = 1



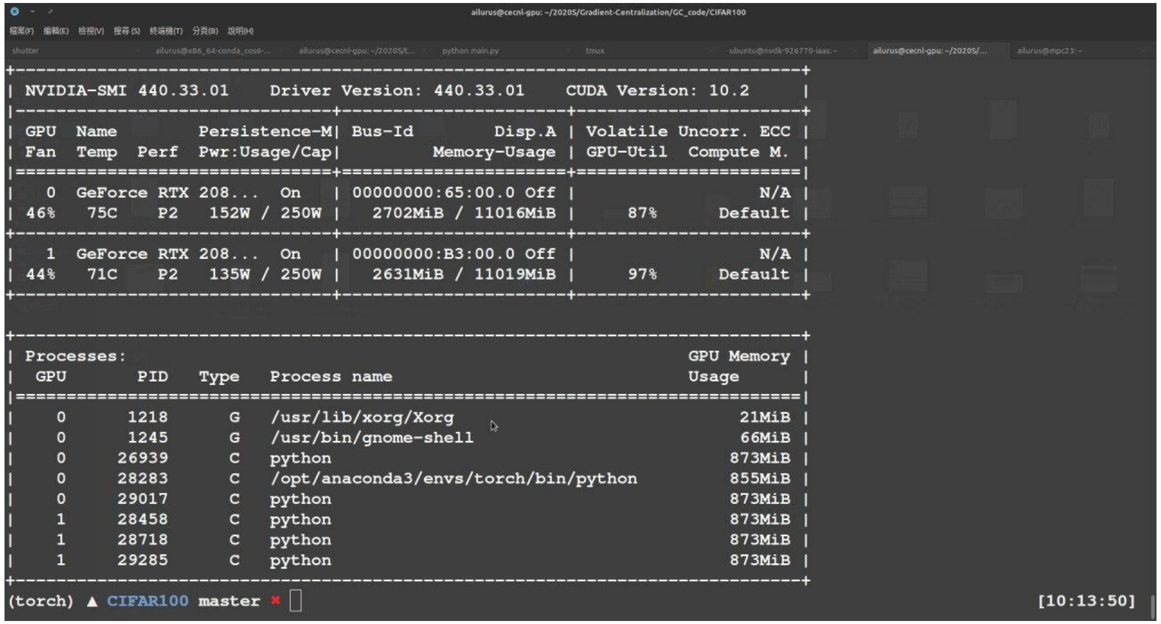
- teacher_forcing ratio = 0.5
- teacher_forcing ratio = 0

other discussion

- This lab is less computational intensive, due to batch_size=1 (I didn't set larger batch_size)



- Hence, I could train many models with different teacher_forcing ratio at the same time



In [ ]:

```

```