0860832 - Thong The Nguyen (阮世聰)

Lab 04 Report

1. Introduction

In this lab, I implement a seq2seq model by modifying encoder, decoder and training funcions, then implement evaluation function and dataloader. Finally, plotting the CrossEntropy training loss, BLEU-4 testing score curves during traning and output correction results from the test.

The highest result from testing data set (from test.json) is 0.8732 BLEU-4 score.

2. Derivation of BPTT

To compute $\nabla_W L$, we observe that:

- The immediate child node of W are all $h^{(t)}$
- The chain rule for tensors can be applied to become

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

To complete the evaluation, we need to know further $\nabla_{h^{(t)}} L$, which can be evaluated using the same chain rule as

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

$$= W^T H^{(t+1)} \left(\nabla_{h^{(t+1)}} L\right) + V^T \left(\nabla_{o^{(t)}} L\right)$$

Where $H^{(t+1)} = \left(\frac{\partial h^{(t+1)}}{\partial a^{(t+1)}}\right)^T$

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

$$\nabla_{o^{(t)}} = \hat{y}^{(t)} - y^{(t)}$$

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

In matrix form, $\nabla_W L$ is given as

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

the gradient of remaining parameters:

$$\nabla_U L = \sum_t H^{(t)} \left(\nabla_{h^{(t)}} L \right) x^{(t)T}$$

$$\nabla_V L = \sum_t \left(\nabla_{o^{(t)}} L \right) h^{(t)T}$$

$$\nabla_b L = \sum_t H^{(t)} \left(\nabla_{h^{(t)}} L \right)$$

$$\nabla_c L = \sum_t \nabla_{o^{(t)}} L$$

3. Implementation details

A. Model implemetation (encoder, decoder, dataloader)

```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(self.hidden_size, self.hidden_size)

    def forward(self, input, hidden):

        output = self.embedding(input)
        output, hidden = self.lstm(output, hidden)
        return output, hidden

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

Encoder

The encoder is implemented using 1 embedding layer to convert the indexed number of each character into a tensor, then feed into LSTM cell for learning the representation of the word.
Note that we need to explicit call *init_tensor*() every time feed a new word into a encoder, otherwise the hidden state will be not attacted to any computational graph, the autograd mechanism will not work after the first run.

```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.lstm = nn.LSTM(hidden_size, hidden_size)
        self.out = nn.Linear(hidden_size, output_size)
        self.softmax = nn.LogSoftmax(dim=1)

    def forward(self, input, hidden):

        output = self.embedding(input)
        output = F.relu(output)
        output, hidden = self.lstm(output, hidden)
        output = self.out(output[0])
        output = self.softmax(output)

        return output, hidden

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

Decoder

Similar to encoder, decoder need a embedding layer to convert index number of character into input tensor, then feeds into LSTM. After that, a Linear and Softmax layer are used to classify the predicted character.

```python
def train(
        input_tensor, target_tensor, encoder, decoder,
        encoder_optimizer, decoder_optimizer, criterion, max_length=MAX_LENGTH
    ):
```

Train function for each word

```
encoder_optimizer.zero_grad()
decoder_optimizer.zero_grad()

input_length = input_tensor.size(0)
target_length = target_tensor.size(0)

encoder_outputs = torch.zeros(max_length, encoder.hidden_size, device=device)

loss = 0
```

Initialization in train function

```
#----------sequence to sequence part for encoder----------#
encoder_hidden = encoder.init_tensor()
encoder_output, encoder_hidden = encoder(input_tensor, encoder_hidden)
```

Encoder part

```
#----------sequence to sequence part for decoder---------#
decoder_hidden = encoder_hidden
decoder_input = torch.tensor([[SOS_token]], device=device)
use_teacher_forcing = True if random.random() < teacher_forcing_ratio else False

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 = torch.tensor([[target_tensor[di]]], device=device)   # Tea

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 = torch.tensor([[topi.squeeze().detach()]], device=device)

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

loss.backward()

encoder_optimizer.step()
decoder_optimizer.step()

return loss.item() / target_length
```

Decoder part and backward

Firstly, we feed an input tensor into encoder, then use the hidden tensor which is outputed from encoder to feed to decoder.

Here, we use 2 methods for learning, which is teacher forcing or not, with ratio 0.5. When using with teacher forcing, we use the ground truth instead of hidden state from last calculation.

Finally, returning the CrossEntropy loss for each training example.

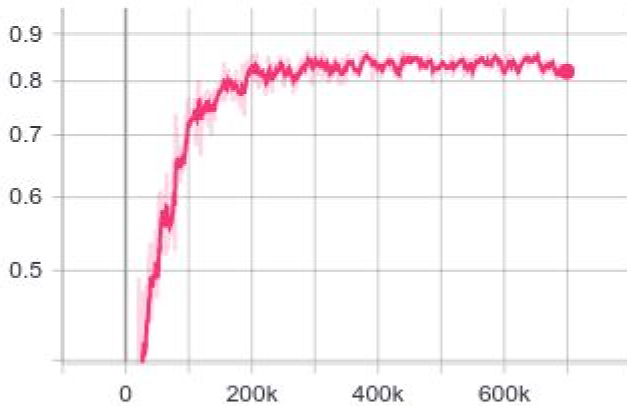
B. Screenshot code of evaluation part

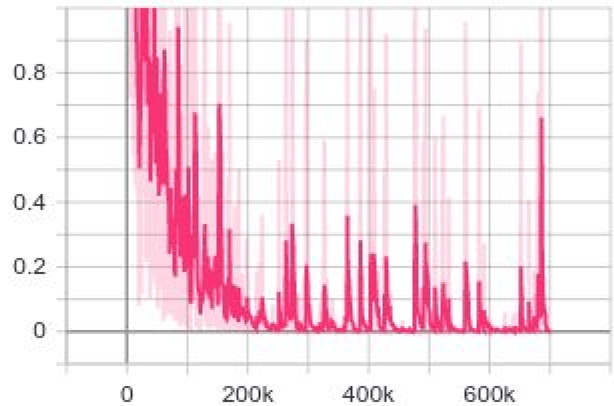
```python
def evaluate(encoder, decoder, filetest, iter, log=True):

    testing_pairs = load_test_data(filetest)
    bleu = 0

    for test_id in range(1, len(testing_pairs) + 1):
        test_pair = testing_pairs[test_id - 1]
        inp = test_pair[0]
        input_tensor = test_pair[1]
        target = test_pair[2]

        encoder_hidden = encoder.init_tensor()
        encoder_output, encoder_hidden = encoder(input_tensor, encoder_hidden)

        decoder_input = torch.tensor([[SOS_token]], device=device)
        decoder_hidden = encoder_hidden

        res = ''
        for di in range(len(target)):
            decoder_output, decoder_hidden = decoder(decoder_input, decoder_hidden)
            topv, topi = decoder_output.topk(1)
            decoder_input = torch.tensor([[topi.squeeze().detach()]], device=device)

            if topi.item() == SOS_token:
                res += '<SOS>'
            elif topi.item() == EOS_token:
                break
            else:
                res += index2char[topi.item()]

        print("==================================")
        print('Input: ', inp)
        print('Predict: ', res)
        print('Target: ', target)

        bleu += compute_bleu(res, target)

    bleu /= len(testing_pairs)
    print('Testing, epoch %d: %f' % (iter, bleu))

    if log:
        writer.add_scalar('BLEU/test', bleu, iter)

    return bleu
```

Screenshot of evalution code that not use ground truth while testing

4. Results and Discussion

Hyperparameters used for training are: hidden size 256, teacher forcing ratio 0.5, learning rate 0.01, training examples 700000.





Average BLEU-4 score on testset (left) and Loss (right) on trainset

The highest average result so far is 0.87 BLEU-4 score on testing dataset.