Deep Learning Practice Lab 4

0856056 Yu-Wun Tseng

April 2020

1 Introduction

The goal of this lab is to implement a seq2seq encoder-decoder network for English spelling correction. The training input are multiple error words, the model need to output the correct words. The encoder and decoder are LSTM, an recurrent neural network is composed of a cell, an input gate, an output gate and a forget gate.

2 Derivation of BPTT

BackPropagation Through Time (BPTT) is used to learn the parameters of recurrent neural networks. In this section, I'll explain the the derivation process. Below is the forward propagation:

$$\begin{aligned} a^{(t)} &= b + Wh^{(t-1)} + Ux^{(t)} \\ h^{(t)} &= tanh(a^{(t)}) \\ o^{(t)} &= c + Vh^{(t)} \\ \hat{y}^{(t)} &= softmax(o^{(t)}) \\ L^{(t)} &= \hat{y}^{(t)} - y^{(t)} \end{aligned} \tag{1}$$

The loss of recurrent neural network is the sum of the loss of all time step, the equation is below:

$$L = \sum_t L^{(t)} \tag{2}$$

First, to calculate the partial derivative of L with respect to $o^{(t)}$:

$$\begin{aligned} \nabla_{o^{(t)}} L &= \frac{\partial L}{\partial \hat{y}^{(t)}} \frac{\partial \hat{y}^{(t)}}{\partial o^{(t)}} \\ &= 1 \cdot softmax'(o^{(t)}) \\ &= \hat{y}^{(t)} - y^{(t)} \end{aligned} \tag{3}$$

As a result, the derivative of the loss with respect to the last hidden state is as below:

$$\nabla_{h^{(\tau)}} L = V^T (y^{(\tau)} - y^{(\tau)}) \tag{4}$$

With the derivative of the last time step, we can use backpropagate to pass the gradient to the first time step. We next calculate $\nabla_{h^{(t)}} L$, the partial derivative of each time step. We need to sum the gradients from two path. One is from $L^{(t+1)}$, the other is from $L^{(t)}$. The equation is as below:

$$\nabla_{h^{(t)}} L = \frac{\partial L}{\partial h^{(t+1)}} \frac{\partial h^{(t+1)}}{\partial h^{(t)}} + \frac{\partial L}{\partial h^{(t)}} \tag{5}$$

The partial derivative of $h^{(t+1)}$ with respect to $h^{(t)}$:

$$\frac{\partial h^{(t+1)}}{\partial h^{(t)}} = \frac{\partial h^{(t+1)}}{\partial a^{(t+1)}} \frac{\partial a^{(t+1)}}{\partial h^{(t)}} = W^T H^{(t+1)} \tag{6}$$

where

$$H^{(t+1)} = \frac{\partial h^{(t+1)}}{\partial a^{(t+1)}}^T \tag{7}$$

The partial derivative of $h^{(t+1)}$ with respect to $h^{(t)}$:

$$\frac{\partial L}{\partial h^{(t)}} = \frac{\partial L}{\partial o^{(t)}} \frac{\partial o^{(t)}}{\partial h^{(t)}} = V^T \frac{\partial L}{\partial o^{(t)}} \tag{8}$$

Thus, $\nabla_{h^{(t)}} L$ will be:

$$\nabla_{h^{(t)}} L = W^T H^{(t+1)} \frac{\partial L}{\partial h^{(t+1)}} + V^T \frac{\partial L}{\partial o^{(t)}} \tag{9}$$

With the result above, we can calculate the gradient of bias and weights. To calculate $\nabla_V L$, the equation is as below:

$$\begin{aligned} \nabla_V L &= \sum_t \frac{\partial L}{\partial o^{(t+1)}} \frac{\partial o^{(t+1)}}{\partial V} \\ &= \sum_t (\nabla_{o^{(t)}} L) h^{(t)} \end{aligned} \tag{10}$$

To calculate $\nabla_U L$, the equation is as below:

$$\begin{aligned} \nabla_U L &= \sum_t \frac{\partial L}{\partial o^{(t+1)}} \frac{\partial o^{(t+1)}}{\partial V} \frac{\partial V}{\partial h^{(t+1)}} \frac{\partial h^{(t+1)}}{\partial U} \\ &= \sum_t (\nabla_{h^{(t)}} L) H^{(t)} x^{(t)} \end{aligned} \tag{11}$$

To calculate $\nabla_W L$, the equation is as below:

$$\begin{aligned} \nabla_W L &= \sum_t \frac{\partial L}{\partial o^{(t+1)}} \frac{\partial o^{(t+1)}}{\partial V} \frac{\partial V}{\partial h^{(t+1)}} \frac{\partial h^{(t+1)}}{\partial h^{(t)}} \\ &= \sum_t (\nabla_{h^{(t)}} L) H^{(t)} h^{(t-1)} \end{aligned} \tag{12}$$

The derivation process of bias is similar to weights. In the end, we can update the weights and bias using the gradient above.

3 Experiment Setups

This lab is done by using Pytorch. The procedure is followed to the official tutorial [1]. However, the official tutorial doesn't consider the batch. Thus, I refer a seq2seq github repository[2] which using batch for training.

3.1 Encoder

The encoder inherits the nn.Module class. Following the tutorial, the encoder consists of an embedding layer and a recurrent neural network. Unlike the tutorial, I use LSTM instead of GRU. According to the LSTM document, the hidden state and the cell memory need to be initialized. Therefore, I modified the initHidden function, using the same size zero tensor to initialize the hidden state and the cell memory. Due to the batch training, I turn the embedded input size to (1, batch_size, hidden_size), which is as same as the hidden state size.

3.2 Decoder

The architecture of the decoder is similar to the encoder, the size of the embedded input, the hidden state and the cell memory are the same as the encoder. Besides the embedding layer and LSTM, the decoder also contains a linear layer and a log softmax layer. After embedding the input, the decoder uses ReLU and passes the output to the LSTM. Finally, softmax the output of the LSTM.

3.3 Dataloader

In order to fetch the input and the target tensor, I implement a dataloader which uses a vocabulary to convert words into tensors. First, I use training data to build a vocabulary, which contains two methods. One is to convert words into vector, the other is to convert vector into words. Next, I split the data with multiple inputs into multiple input-target pairs. Finally, I implement the ___getitem___ function and convert the words into tensor. Besides, I shuffle the training data.

```python
    #----------sequence to sequence part for encoder----------#
    for ei in range(input_length):
        encoder_output, encoder_hidden = encoder(
            input_tensor[ei], encoder_hidden)

    decoder_input = torch.tensor([SOS_token for i in range(batch_size)], device=device)
    output = torch.zeros(target_length, batch_size)

    decoder_hidden = encoder_hidden

    #----------sequence to sequence part for decoder----------#
    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
        output[di] = decoder_input

    # get predict indices
    output = output.transpose(0, 1)

    # convert indices into string
    for idx in range(batch_size):
        prediction.append(vocab.indices2word(output[idx].data.numpy()))

# calculate average BLEU score
avg_bleu = show_prediction(inputs, prediction, targets, plot_pred)
```

(a) Generate the prediction

```python
#compute BLEU-4 score
def compute_bleu(output, reference):
    cc = SmoothingFunction()
    if len(reference) == 3:
        weights = (0.33,0.33,0.33)
    else:
        weights = (0.25,0.25,0.25,0.25)
    return sentence_bleu([reference], output,weights=weights,smoothing_function=cc.method1)
```

(b) Compute the BLEU score

Figure 1: Code of evaluation

3.4 Training

The train function and trainIter function follow the tutorial, except the input and target tensor parts. In my implementation, I put a batch of input tensor and target tensot into train function. Due to the LSTM specification, the dimension is (seq_len, batch_size, hidden_size), thus, I transpose the tensor to fit the specification. At each time step, the input and output of the encoder and decoder is a batch containing only one character.

I set the teaching forcing ratio to 0.8. If using the teacher forcing, the input of the decoder is the target character instead of the previous output of the decoder. The learning rate is 0.01, the batch size is 32 and the hidden size is 256.

The prediction are generated as the Figure 1a shows, and the computation of the BLEU score are shown as Figure 1b.

4 Results

4.1 Loss curve

The loss curve is shown on Figure 2. The loss has fallen to half in first twenty epochs and became stable after forty epochs.

4.2 BLEU score curve

The BLEU score curve is shown on Figure 3. The score increased rapidly in the first thirty epochs and continued to increase steadily.

4.3 Predict result

As the Figure 4 and Figure 5 show, the model can predict most inputs correctly.

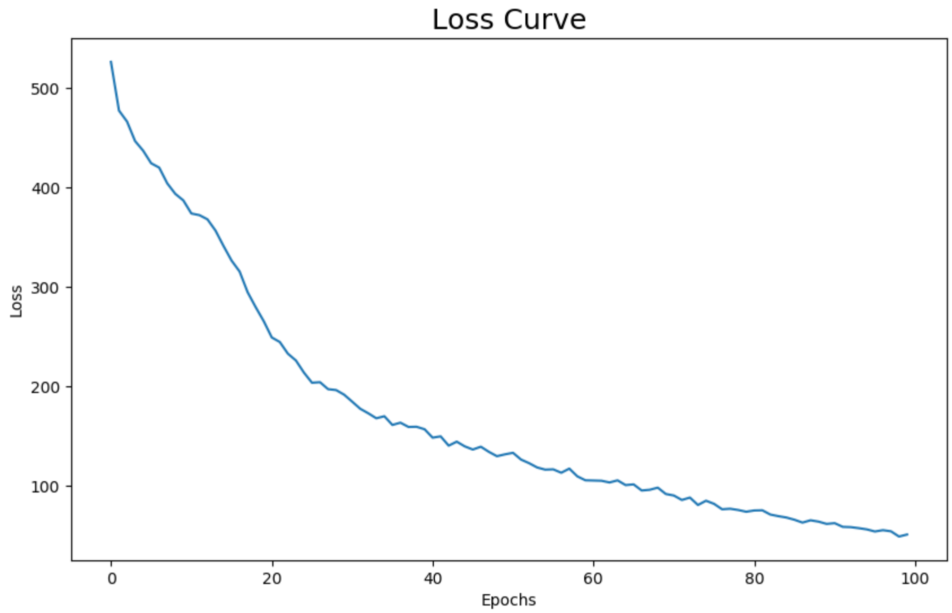


Figure 2: Loss trend

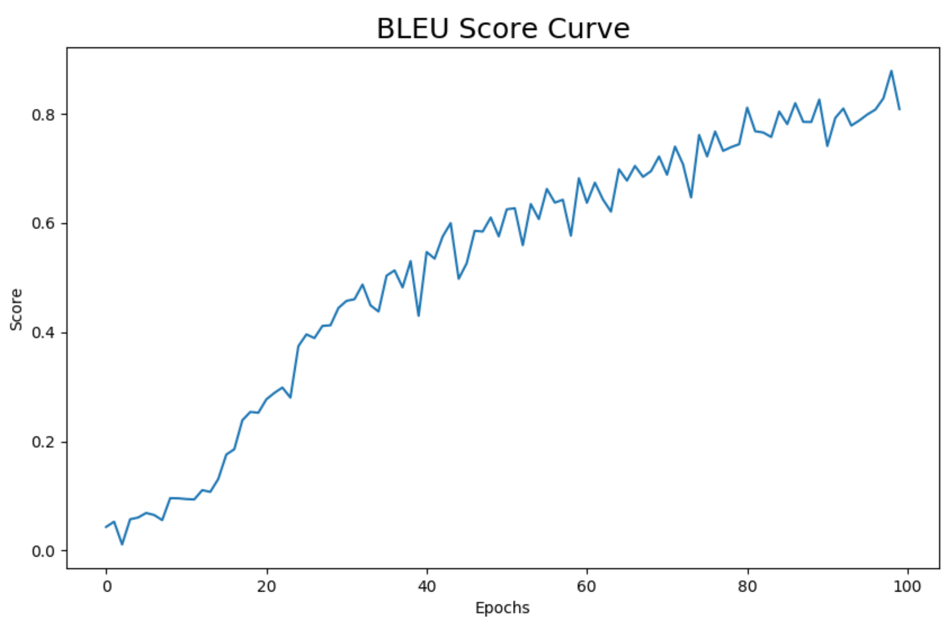
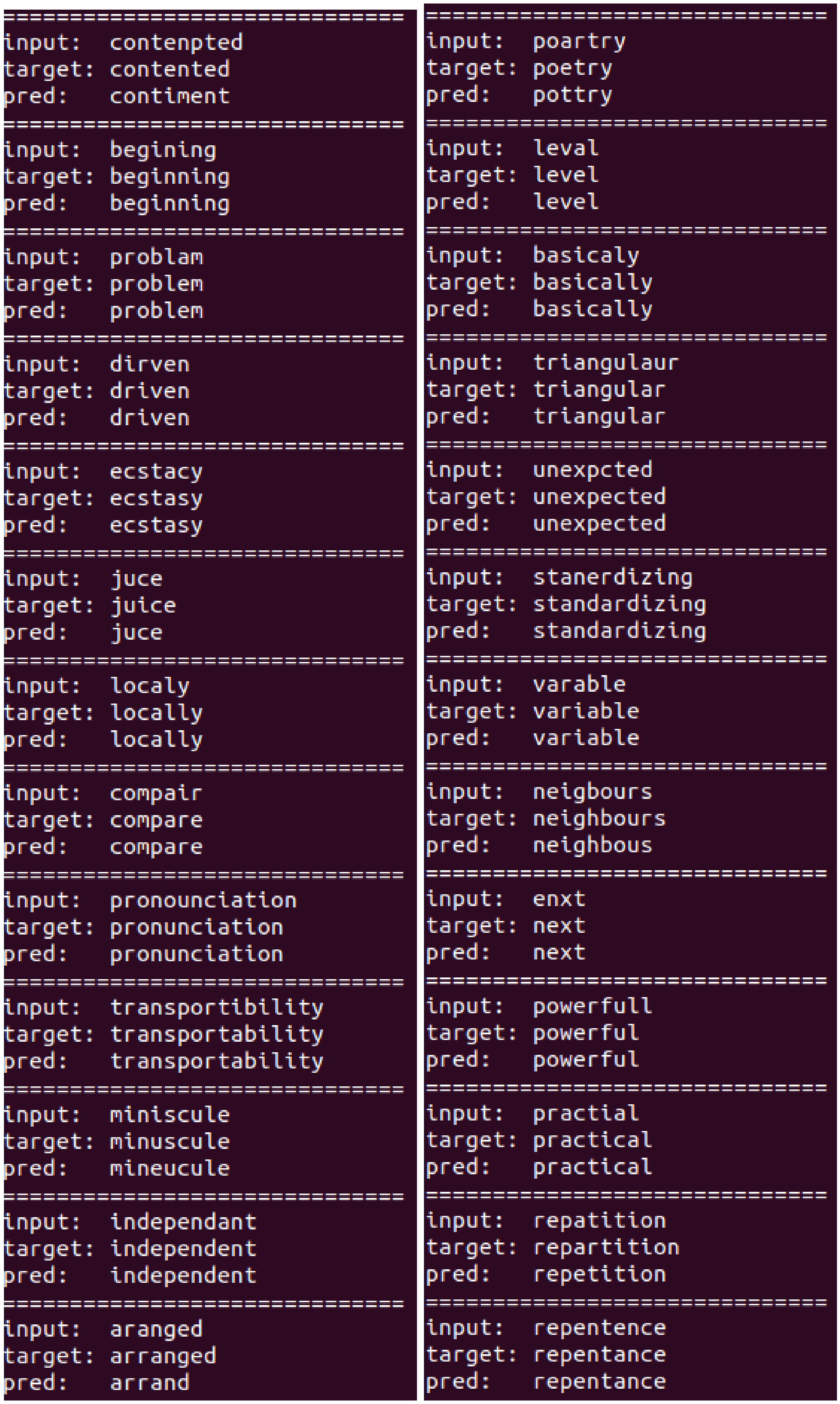


Figure 3: BLEU score curve



(a) (b)

Figure 4: Prediction

```
===============================
input:  substracts
target: subtracts
pred:   subtracts
===============================
input:  beed
target: bead
pred:   bead
===============================
input:  beame
target: beam
pred:   beam
===============================
input:  decieve
target: deceive
pred:   deceive
===============================
input:  decant
target: decent
pred:   decent
===============================
input:  dag
target: dog
pred:   dog
===============================
input:  daing
target: doing
pred:   doing
===============================
input:  expence
target: expense
pred:   expense
===============================
input:  feirce
target: fierce
pred:   fierce
===============================
input:  firery
target: fiery
pred:   fiery
===============================
input:  fought
target: fort
pred:   fort
===============================
input:  fourth
target: forth
pred:   forth
===============================
input:  ham
target: harm
pred:   ham
```

(a)

```
===============================
input:  havest
target: harvest
pred:   harvest
===============================
input:  immdiately
target: immediately
pred:   immediately
===============================
input:  inehaustible
target: inexhaustible
pred:   ineghaustible
===============================
input:  journel
target: journal
pred:   journal
===============================
input:  leason
target: lesson
pred:   lesson
===============================
input:  mantain
target: maintain
pred:   mantian
===============================
input:  miricle
target: miracle
pred:   miracle
===============================
input:  oportunity
target: opportunity
pred:   opportunity
===============================
input:  parenthasis
target: parenthesis
pred:   parenthesis
===============================
input:  recetion
target: recession
pred:   recestion
===============================
input:  scadual
target: schedule
pred:   schedule
BLEU-4 score: 0.8785
```

(b)

Figure 5: Prediction

5 Discussion

The tutorial doesn't consider the batch training, the input is an input-target pair picked up randomly. Both of the input sequence length of each time step and the batch size are 1. However, if I want to use batch, I need to get the corresponding characters of each input in the batch. Thus, I need to transpose the input tensor and get the raw data. In addition, the LSTM class of Pytorch is not batch first if I don't change the default setting. Therefore, I need to ensure that all inputs and parameters have right dimensions. Otherwise, the model will break.

In the beginning, my model couldn't learn to predict correctly. No matter how much epochs I used, the performance was poor. Although my model was three times faster than my classmate's model, my score was one third of hers. Then I found that if I didn't use the teacher forcing, the decoder would break the loop if the model predicted the EOS token. The model had not been learned well at the beginning, and the vocabulary was small. As a result, the model was prone to predict the EOS token and end the loop prematurely. This lead to poor model learning. After I deleted the early break part, the model learned well.

References

[1] Sean Robertson. Nlp from scratch: Translation with a sequence to sequence network and attention.

[2] pengyuchen. Pytorch-batch-seq2seq.