South China University of Technology

The Experiment Report of *Machine Learning*

SCHOOL: SCHOOL OF SOFTWARE ENGINEERING

SUBJECT: SOFTWARE ENGINEERING

Author:
Bin Li
Liandong Huang
Jinqiang Liu

Supervisor:
Mingkui Tan

Student ID:
201836661148
201830660499
201830680169p

Grade:
2018 undergraduate

March 1, 2021

Chinese-English Translation Machine Based on Sequence to Sequence Network

***Abstract*—The Seq2Seq model is the most important variant of RNN. This structure is also called Encoder-Decoder model.**

I. Introduction

THE Seq2seq belongs to a kind of encoder-decoder structure. The basic idea of the common encoder-decoder structure is to use two RNNs, one RNN as the encoder and the other RNN as the decoder. The encoder is responsible for compressing the input sequence into a vector of a specified length. This vector can be regarded as the semantics of the sequence. This process is called encoding. The easiest way to obtain the semantic vector is to directly use the hidden state of the last input as the semantic vector C . It is also possible to perform a transformation on the last hidden state to obtain the semantic vector, or to perform a transformation on all the hidden states of the input sequence to obtain the semantic variable.

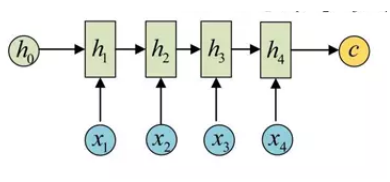


Fig. 1. RNN network

The decoder is responsible for generating the specified sequence based on the semantic vector. This process is also called decoding, as shown in the figure below. The simplest way is to input the semantic variable obtained by the encoder as the initial state into the RNN of the decoder to obtain the output sequence. It can be seen that the output at the previous moment will be used as the input at the current moment, and the semantic vector C is only used as the initial state to participate in the operation, and the subsequent operations have nothing to do with the semantic vector C.

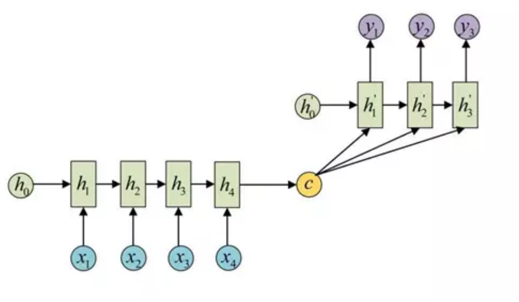


Fig. 2. Semantic vector participates in every process of decoding

II. Methods and Theory

A. The Encoder

The encoder of a seq2seq network is a RNN that outputs some value for every word from the input sentence. For every input word the encoder outputs a vector and a hidden state, and uses the hidden state for the next input word.

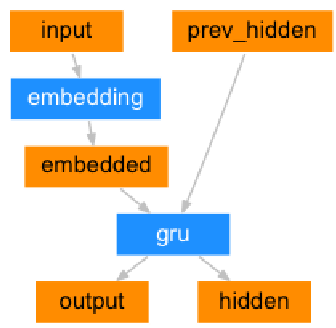


Fig. 3. Encoder Network

B. The Attention Decoder

If only the context vector is passed between the encoder and decoder, that single vector carries the burden of encoding the entire sentence.

Attention allows the decoder network to focus on a different part of the encoders outputs for every step of the decoders own outputs. First we calculate a set of attention weights. These will be multiplied by the encoder output vectors to create a weighted combination. The result (called attn_applied in the code) should contain information about that specific part of the input sequence, and thus help the decoder choose the right output words.

Calculating the attention weights is done with another feed-forward layer attn, using the decoders input and hidden state as inputs. Because there are sentences of all sizes in the training data, to actually create and train this layer we have to choose a maximum sentence length (input length, for encoder outputs) that it can apply to. Sentences of the maximum length will use all the attention weights, while shorter sentences will only use the first few.

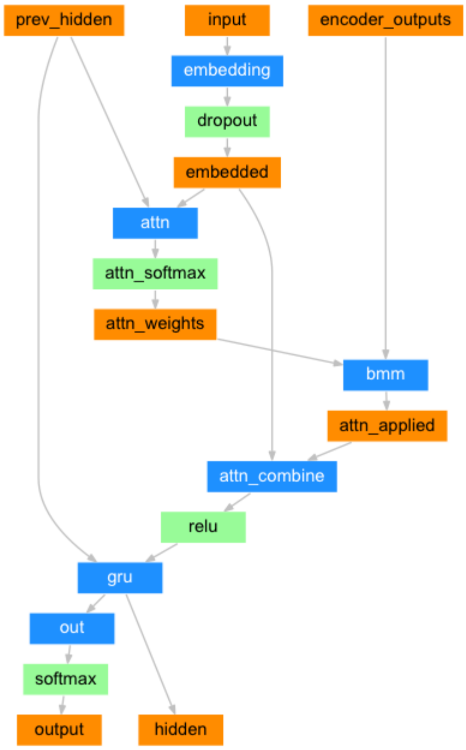


Fig. 4. Attention Decoder Network

III. Experiments

A. Dataset

Use Chinese and English translation datasetmore translation data sets can be downloaded on this website.

There are a total of 23,610 translation data pairs, and each pair of translation data is on the same line: English on the left, Chinese in the middle, and other attributes information on the right. The separator is tab.

B. Implementation

Loading data files

Similar to the character encoding used in the character-level RNN tutorials, we will be representing each word in a language as a one-hot vector, or giant vector of zeros except for a single one (at the index of the word). Compared to the dozens of characters that might exist in a language, there are many many more words, so the encoding vector is much larger. We will however cheat a bit and trim the data to only use a few thousand words per language.

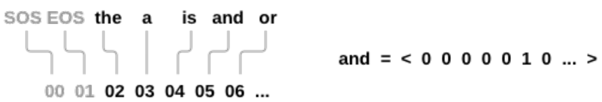


Fig. 5. word encoding

The data for this project is a set of many thousands of Chinese to English translation pairs. Download Chinese-English translation dataset.

Read the dataset by row and remove the attribute information (only use top-2 split for each line) when constructing the training data pair, otherwise an error will be reported.

Split words from the training sentences and construct a comparison table of Chinese and English words in the dataset.

Build a machine translation model

Chinese-English translation machine is based on sequence to sequence network

1. Build the encoder(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.gru=nn.GRU(hidden_size,
        hidden_size)
  def forward(self,input,hidden):
    embedded=self.embedding(input).view
        (1,1,-1)
    output=embedded
    output,hidden=self.gru(output,
        hidden)
    return output,hidden
  def initHidden(self):
    return torch.zeros(1,1,self.
        hidden_size,device=device)
```

2. Build a decoder based on the attention mechanism(Attention Decoder).

```
class AttnDecoderRNN(nn.Module):
  def _init_(self,hidden_size,
      output_size,dropout_p=0.1,
      max_length=MAX_LENGTH):
    super(AttnDecoderRNN,self)._init_()
    self.hidden_size=hidden_size
    self.output_size=output_size
    self.dropout_p=dropout_p
    self.max_length=max_length
    self.embedding=nn.Embedding(self.
        output_size,self.hidden_size)
    self.attn=nn.Linear(self.hidden_size
        *2,self.max_length)
    self.attn_combine=nn.Linear(self.
        hidden_size*2,self.hidden_size)
    self.dropout=nn.Dropout(self.
        dropout_p)
    self.gru=nn.GRU(self.hidden_size,
        self.hidden_size)
    self.out=nn.Linear(self.hidden_size,
        self.output_size)
  def forward(self,input,hidden,
      encoder_outputs):
    embedded=self.embedding(input).view
```

```
            (1,1,-1)
        embedded=self.dropout(embedded)
        attn_weights=F.softmax(
            self.attn(torch.cat((embedded[0],
                hidden[0]),1)),dim=1)
        attn_applied=torch.bmm(attn_weights.
            unsqueeze(0),
            encoder_outputs.unsqueeze(0))
        output=torch.cat((embedded[0],
            attn_applied[0]),1)
        output=self.attn_combine(output).
            unsqueeze(0)
        output=F.relu(output)
        output, hidden = self.gru(output,
            hidden)
        output=F.log_softmax(self.out(output
            [0]), dim=1)
        return output,hidden,attn_weights
    def initHidden(self):
        return torch.zeros(1,1,self.
            hidden_size,device=device)
```

Train machine translation model

The whole training process looks like this:

- Start a timer
- Initialize optimizers and criterion
- Create set of training pairs
- Then we call train many times and occasionally print the progress

Then we call train many times and occasionally print the progress (% of examples, time so far, estimated time) and average loss.

```
hidden_size = 256
encoder1 = EncoderRNN(input_lang.n_words, hidden_size).to(device)
attn_decoder1 = AttnDecoderRNN(hidden_size, output_lang.n_words, dropout_p=0.1).to(device)

plot_losses = trainIters(encoder1, attn_decoder1, 75000, print_every=5000)
```

```
1m 52s (- 26m 13s) (5000 6%) 2.1492
3m 49s (- 24m 50s) (10000 13%) 0.4847
5m 46s (- 23m 6s) (15000 20%) 0.0756
7m 43s (- 21m 15s) (20000 26%) 0.0526
9m 41s (- 19m 22s) (25000 33%) 0.0408
11m 38s (- 17m 28s) (30000 40%) 0.0388
13m 35s (- 15m 32s) (35000 46%) 0.0324
15m 32s (- 13m 35s) (40000 53%) 0.0340
17m 28s (- 11m 39s) (45000 60%) 0.0359
19m 24s (- 9m 42s) (50000 66%) 0.0351
21m 20s (- 7m 45s) (55000 73%) 0.0350
23m 17s (- 5m 49s) (60000 80%) 0.0305
25m 16s (- 3m 53s) (65000 86%) 0.0323
27m 14s (- 1m 56s) (70000 93%) 0.0292
29m 11s (- 0m 0s) (75000 100%) 0.0295
```

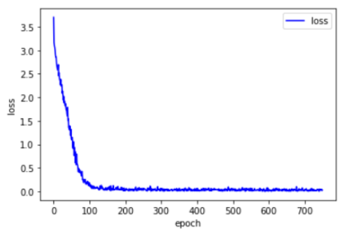


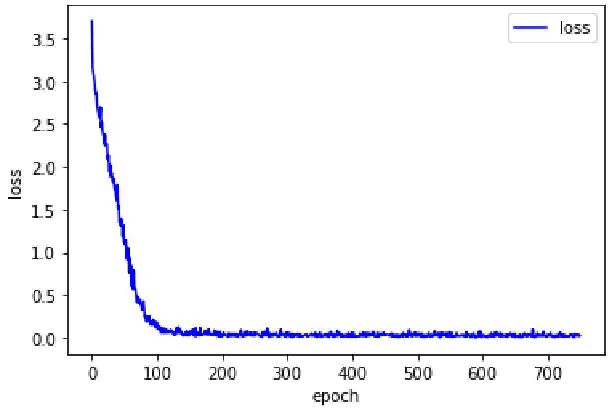
Fig. 6. Training process

At last we evaluate the trained model by BLEU (The full name is Bilingual Evaluation Understudy). More details can be found in Python Natural Language Toolkit Library (NLTK).

IV. Results

Loss value

The graph of loss value of the translation model varying with the number of iterations:



Translation results

We evaluate 100 random sentences from the training set and print out the input, target, and output and calculate the average BLEU score to evaluate the trained model.

```
> 他 是 一 個 數 學 天 才 。
= he is a mathematical genius .
< he is a mathematical genius . <EOS>
bleu score =  1.0

> 他 是 个 棒 球 手 。
= he is a baseball player .
< he is a baseball player . <EOS>
bleu score =  1.0

> 他 一 天 一 天 地 好 轉 。
= he is getting better day by day .
< he is getting better day by day . <EOS>
bleu score =  1.0

> 她 现 在 正 在 工 作 。
= she is at work right now .
< she is at work right now . <EOS>
bleu score =  1.0

> 我 是 一 名 電 工 。
= i am an electrician .
< i am an electrician . <EOS>
bleu score =  1.0

> 她 是 个 倔 强 的 女 孩 。
= she is an obstinate girl .
< she is an obstinate girl . <EOS>
bleu score =  1.0

> 他 是 个 棒 球 手 。
= he is a baseball player .
< he is a baseball player . <EOS>
bleu score =  1.0

average bleu score =  0.9763894310424627
```

Fig. 7. the average BLEU score of 100 sentences is 0.976

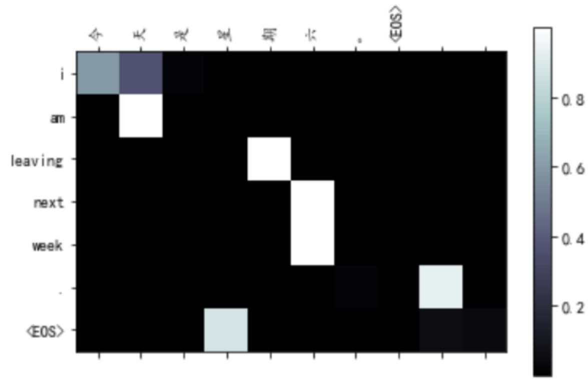
Visualizing Attention

A useful property of the attention mechanism is its highly interpretable outputs. Because it is used to weight specific encoder outputs of the input sequence, we can imagine looking where the network is focused most at each time step.

we use matplotlib to see attention output displayed as a matrix, with the columns being input steps and rows being output steps:

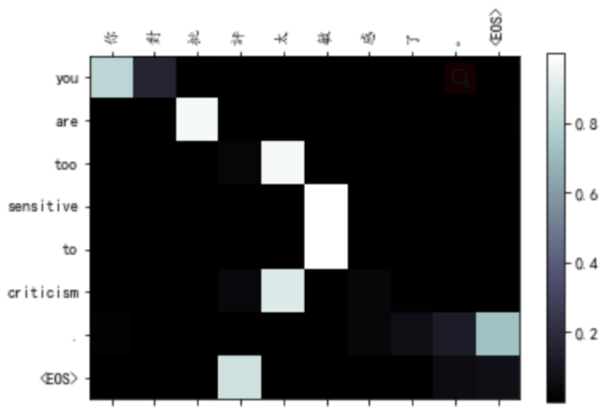
```
evaluateAndShowAttention("今 天 是 星 期 六 。")
```

```
input = 今 天 是 星 期 六 。
output = i am leaving next week . <EOS>
```




```
evaluateAndShowAttention("你 對 批 評 太 敏 感 了 。")
```

```
input = 你 對 批 評 太 敏 感 了 。
output = you are too sensitive to criticism . <EOS>
```



V. Conclusion

Sequence-to-sequence (seq2seq) model, which is a model with significant effects in natural language processing technology. seq2seq breaks through the traditional fixed-size input problem framework, and it also opens the door for applying classic deep neural network models to sequential tasks such as translation and functional Q&A.

In this experiment, Due to the small number of experimental data sets, the trained translation model does not perform well on certain sentences. But on most of sentences, the translation model still gives good results.