Final Term Project: Chinese-To-English Machine Translation

20337021 陈浚铭

Model: Seq2Seq with Bahdanau attention

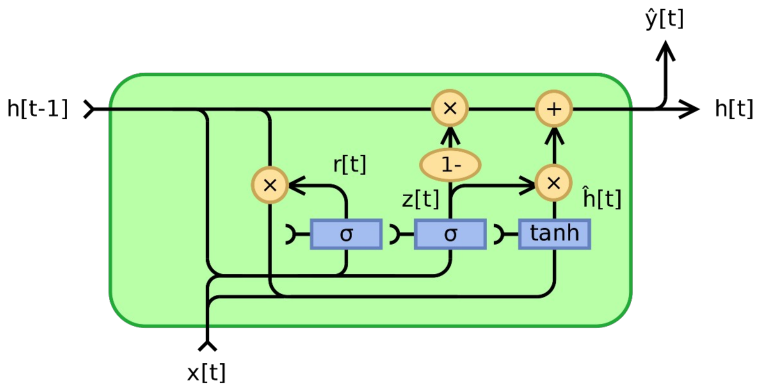
Underlying architecture: GRU

We make use of Gated recurrent units(GRUs) for the seq2seq model. GRU is like a long short-term emmory(LSTM) with a forget gate, but has fewer parameters than LSTM, as it lacks an output gate.

Fully gated unit

Initially for t = 0, the output vector $h_0 = 0$

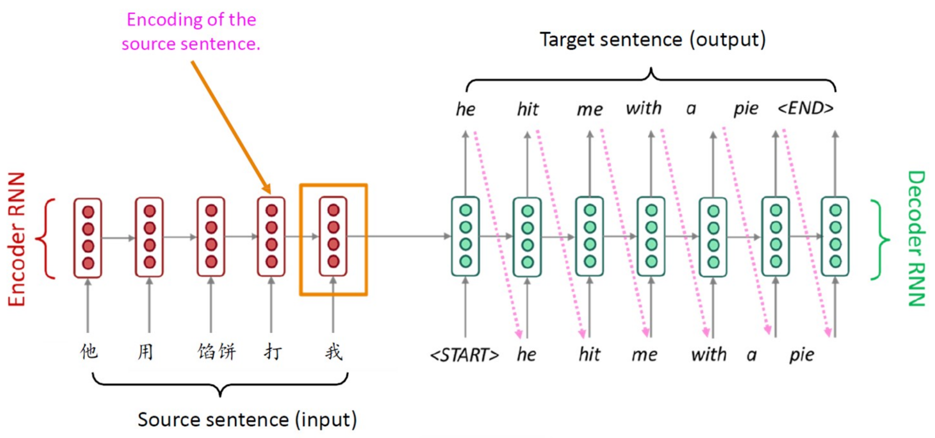
$$z_t = \sigma(W_z X_t + U_z h_{t-1} + b_z) \quad (1)$$
$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r) \quad (2)$$
$$\hat{h}_t = \tanh(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h) \quad (3)$$
$$h_t = (1 - z_t) \odot h_{t+1} + z_t \odot \hat{h}_t \quad (4)$$



Variables:
x_t: input variable
h_t: output variable
$\hat{h}_t$: candidate activation
vector z_t: update gate vector
r_t: reset gate vector
W, U and b: parameter, matrices and vector
Activation functions : logistic function, tanh: hyperbolic function
in contrast to [Bahdanau et al.] that make use of a bidirectional RNN model as encoder, we make use of a GRU as encoder and GRU as decoder for the seq2seq model with Bahdanau attention.

Why use Attention for Seq2Seq?

Consider diagram mentioned from lecture 10,


Encoding of the source sentence.
Target sentence (output)
he hit me with a pie <END>
Encoder RNN
Decoder RNN
他 用 馅饼 打 我
<START> he hit me with a pie
Source sentence (input)

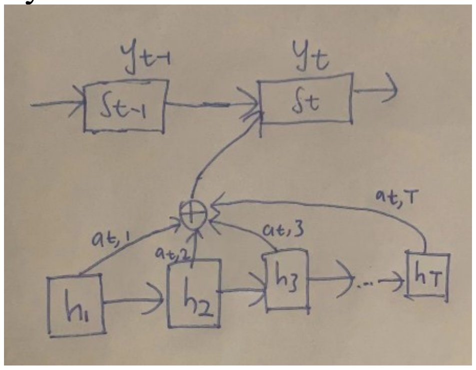
With this basic encoder-decoder model, the decoder takes as input the encoding of the source sentence, which captures all the information about the source sentence.The seq2seq model will work well for small to medium size sentences, but for large sentences, the effectiveness degrades, as the context of the first part of the sentence will have very little effect on the final part vector passed to the decoder. In order to resolve the information bottleneck problem. We can use the attention mechanism: on each step of the decoder, use direct connection to "focus" on a particular part of the source sequence.

The bahdanau architecture:

The main components of the Bahdanau architecture are the following:
. s_{t-1}: hidden decoder state at the previous time step t-1.
. c_t is the context vector at time step t. It is uniquely generated at each decoder step to generate a target word, y_t
. h_i is the annotation that captures the information contained in the words forming the entire input sentence, $\{x_1. x_2, \ldots, x_T\}$, with strong focus around the i-th word out of T total words.
. $a_{t,i}$ is the weight value assigned to each annotation h_i, at the current time step t.
. $e_{t,i}$ is the attention score generated by an alignment model, a(.), that scores how well s_{t-1} and h_i match.

Encoder:

The role of the encoder is to generate an annotation h_i, for every word, x_i, in an input sentence of length T words. The annotation list $h = [h_1, h_2, \ldots, h_T]$ thus is simply the encoder outputs, and the hidden layer.



Decoder:

The role of the decoder is to produce the target words by focusing on the most relevant information contained in the source by using the **attention mechanism**.

Given that the decoder has predicted the sentence part $y_2, y_2, \ldots y_{t-1}$ and computed context vector c_t, the decoder is trained to predict the next word y_t. the decoder thus defines a probability over translation y by decomposing the joint probability into the ordered conditionals.

$$p(y) = \prod_{t=1}^{T} p(y_t|y_1, \dots, y_{t-1}, x)$$

dene each condition probability

$$p(y_t|y_1, \dots, y_{t-1} = g(y_{i-1}, s_i, c_i)$$

where s_i is the RNN hidden state for time i, computed by

$$s_i = f(s_i, y_{t-1}, c_i)$$

The decoder takes each annotation and feeds it to an alignment model, a(.), together with the previous decoder state s_{t-1}. This generates an attention score.

$$e_{t,i} = v_a^T \tanh(W_a s_{t-1}) + U_a h_i$$

Then, the attention score could be used to calculuate the attention weight value
$a_{t,i}$ = softmax($e_{t,i}$)
The application of the softmax function normalizes the annotation value to range between 0 and 1, hence, the resulting weights are probability values. Each probability value reflects how important h_i or s_i are in generating the next state s_t and next output y_t
Finally, the context vector can be calculated, which is the weighted sum of the annotations:

$$c_t = \Sigma_{i=1}^{T} a_{t,i} h_i$$

The Bahdanau Attention Algorithm:

In summary, the attention algorithm proposed by [Bahdanau et al]. performs the following operations:

1.The encoder generates a set of annotations, h_i, from the input sentence.
2.These annotations are fed to an alignment model and the previous hidden decoder state. The alignment model uses this information to generate the attention scores, **$e_{t,i}$**
3.A softmax function is applied to the attention scores, effectively normalizing them into weight values, **$a_{t,i}$**, in a range between 0 and 1.

4.Together with the previously computed annotations, these weights are used to generate a context vector,c_t, through a weighted sum of the annotations.
5.The context vector is fed to the decoder together with the previous hidden decoder state and the previous output to compute the final output,y_t.
6.Steps 2-6 are repeated until the end of the sequence.

重点代码：

seq2seq_model.py

Encoder 类

```python
class EncoderRNN(nn.Module):
    def __init__(self, input_size, hidden_size, dropout_p=0.1):
        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, batch_first=True)#, bidirectional = True)
        self.dropout = nn.Dropout(dropout_p)

    def forward(self, input):
        embedded = self.dropout(self.embedding(input))
        output, hidden = self.gru(embedded)
        return output, hidden
```

Attention 类（负责计算 context vector 以递给 Decoder)

```python
class BahdanauAttention(nn.Module):
    def __init__(self, hidden_size):
        super(BahdanauAttention, self).__init__()
        self.Wa = nn.Linear(hidden_size, hidden_size)
        self.Ua = nn.Linear(hidden_size, hidden_size)
        self.Va = nn.Linear(hidden_size, 1)

    def forward(self, query, keys):
        scores = self.Va(torch.tanh(self.Wa(query) + self.Ua(keys)))
        scores = scores.squeeze(2).unsqueeze(1)  #attention score

        weights = F.softmax(scores, dim=-1) # compute a_t,i
        context = torch.bmm(weights, keys) # compute the context vector c_t

        return context, weights
```

Decoder 类

```
class AttnDecoderRNN(nn.Module):
    def __init__(self, hidden_size, output_size, dropout_p=0.1):
        super(AttnDecoderRNN, self).__init__()
        self.embedding = nn.Embedding(output_size, hidden_size)
        self.attention = BahdanauAttention(hidden_size)
        self.gru = nn.GRU(2 * hidden_size, hidden_size, batch_first=True)
        self.out = nn.Linear(hidden_size, output_size)
        self.dropout = nn.Dropout(dropout_p)

    def forward(self, encoder_outputs, encoder_hidden, target_tensor=None):
        batch_size = encoder_outputs.size(0)
        decoder_input = torch.empty(batch_size, 1, dtype=torch.long, device=device).fill_(SOS_token)
        decoder_hidden = encoder_hidden
        decoder_outputs = []
        attentions = []

        for i in range(MAX_LENGTH):
            decoder_output, decoder_hidden, attn_weights = self.forward_step(
                decoder_input, decoder_hidden, encoder_outputs
            )
            decoder_outputs.append(decoder_output)
            attentions.append(attn_weights)

            if target_tensor is not None:
                # teacher forcing
                decoder_input = target_tensor[:, i].unsqueeze(1) # Teacher forcing
            else:
                _, topi = decoder_output.topk(1)
                decoder_input = topi.squeeze(-1).detach()  # detach from history as input

        decoder_outputs = torch.cat(decoder_outputs, dim=1)
        decoder_outputs = F.log_softmax(decoder_outputs, dim=-1)
        attentions = torch.cat(attentions, dim=1)

        return decoder_outputs, decoder_hidden, attentions
```

注意到我们在训练过程中使用了 teacher forcing。这个方法的优点在于收敛速度快，且模型稳定，但缺点是它没有根据自己的错误来训练。另外，我们定义的 forward 函数在 for 循环调用了中 forward_step

```
def forward_step(self, input, hidden, encoder_outputs):
    embedded =  self.dropout(self.embedding(input))

    query = hidden.permute(1, 0, 2)
    context, attn_weights = self.attention(query, encoder_outputs) # calculate context vector
    input_gru = torch.cat((embedded, context), dim=2) # compute using context

    output, hidden = self.gru(input_gru, hidden)
    output = self.out(output)

    return output, hidden, attn_weights
```

在 forward_step 函数，可以看到使用 context vector 和隐藏层作为 gru 模型的输入(input_gru)。

之后，在 train_eval.py 负责生成词库，训练和验证数据集，test.py 负责测试数据

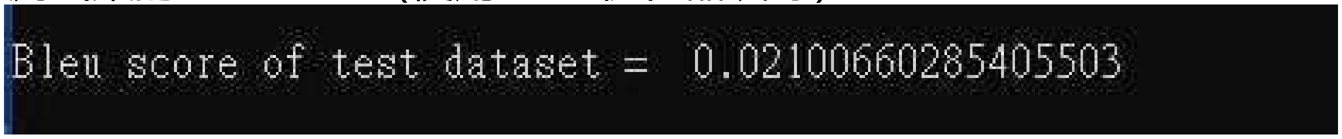
实验训练内容

在数据预处理部分，我们过滤了超过长度为 35 的句子，当中对于中文使用 jieba, 英文使用 nltk 进行分词。

在测试和验证数据方面，使用了 batch=16 并分别使用 SGD 和 Adam 优化器来进行训练。我们训练 100 次，每 5 次我们通过 bleu score 进行验证数据，并保存最好的模型。

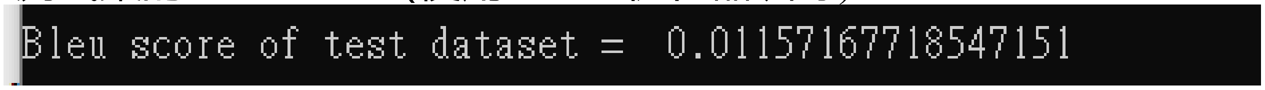
实验测试结果：

测试集的 **bleu score**（使用 SGD 优化器训练）：



```
Bleu score of test dataset =  0.02100660285405503
```

测试集的 **bleu score**（使用 Adam 优化器训练):



```
Bleu score of test dataset =  0.01157167718547151
```

实验心得：

我觉得这个 project 能够让我理解机器翻译背后使用神经网络的理论和实现。机器翻译是一个很伟大的工具，让我们能够跨越语言的边界来跟不同语言的人沟通。我在研究 Bahdanau attention 的论文能够让我理解 attention 的实现方法，和应用到 seq2seq 模型的工作原理，好处等等。在代码实现方面让我提升了我的能力，激发我对人工智能的兴趣。