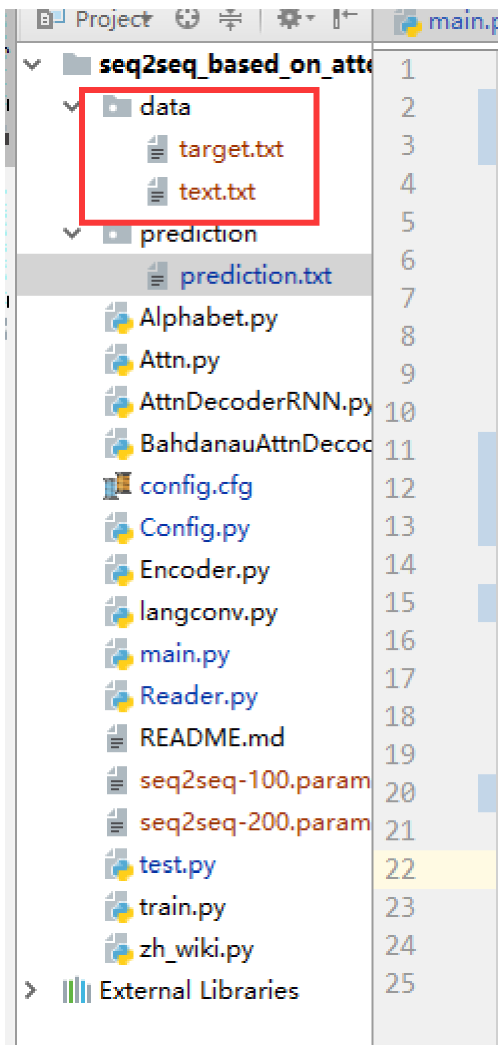
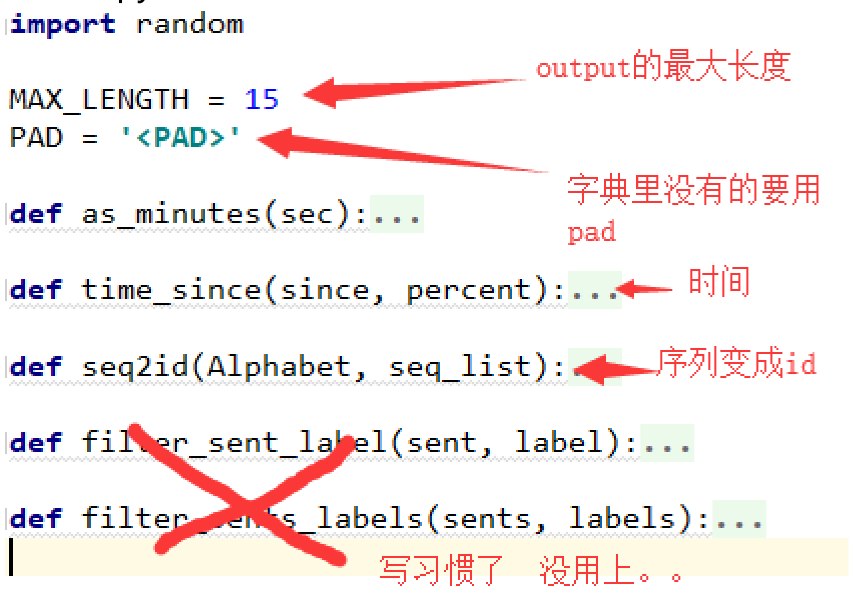
按照顺序说明：

1 你给的是文本是没分的，我自己写了小脚本分了下，最终的text和target文件在：



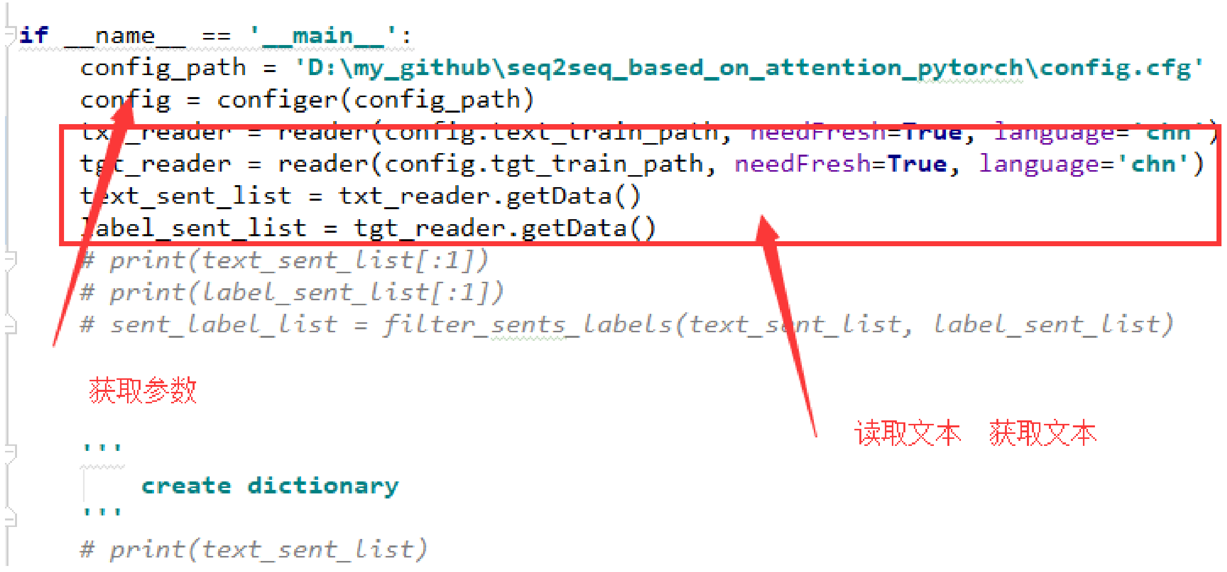
```
Project
seq2seq_based_on_atte
  data
    target.txt
    text.txt
  prediction
    prediction.txt
  Alphabet.py
  Attn.py
  AttnDecoderRNN.py
  BahdanauAttnDecoc
  config.cfg
  Config.py
  Encoder.py
  langconv.py
  main.py
  Reader.py
  README.md
  seq2seq-100.param
  seq2seq-200.param
  test.py
  train.py
  zh_wiki.py
External Libraries
```

2.main.py是主要流程所在。



```
import random

MAX_LENGTH = 15
PAD = '<PAD>'

def as_minutes(sec):...

def time_since(since, percent):...

def seq2id(Alphabet, seq_list):...

~~def filter_sent_label(sent, label):...~~

~~def filter_sents_labels(sents, labels):...~~
```

接下来进入__main__:

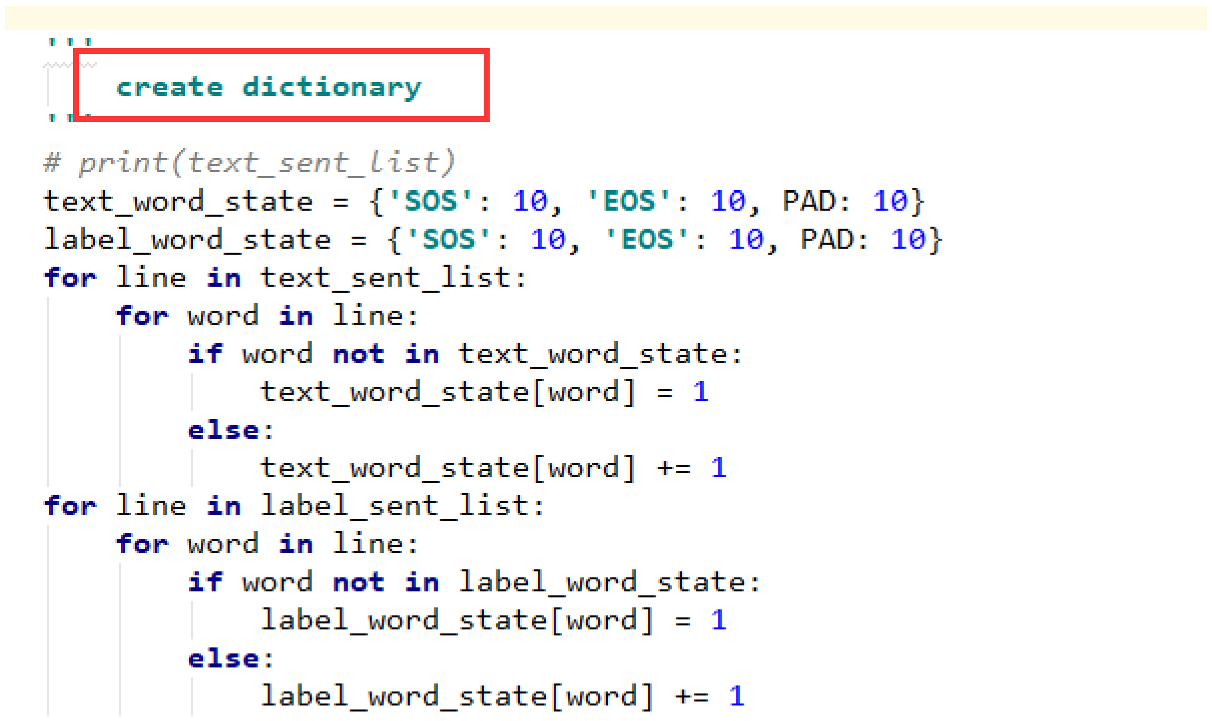


```
if __name__ == '__main__':
    config_path = 'D:\my_github\seq2seq_based_on_attention_pytorch\config.cfg'
    config = configer(config_path)
    txt_reader = reader(config.text_train_path, needFresh=True, language='chn')
    tgt_reader = reader(config.tgt_train_path, needFresh=True, language='chn')
    text_sent_list = txt_reader.getData()
    label_sent_list = tgt_reader.getData()
    # print(text_sent_list[:1])
    # print(label_sent_list[:1])
    # sent_label_list = filter_sents_labels(text_sent_list, label_sent_list)

    '''
        create dictionary
    '''
    # print(text_sent_list)
```

config就不用说了，很简单的。

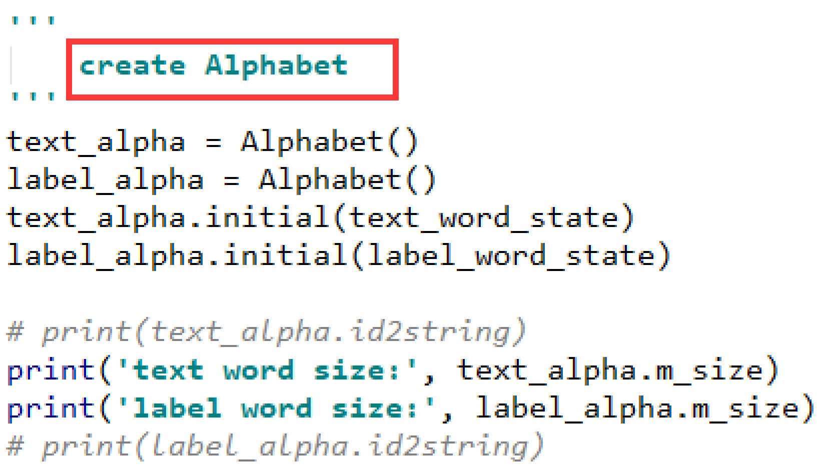
读取文本 就是清洗下中文的。我Reader里有注释，可以看看。其实你也懂。

建立字典

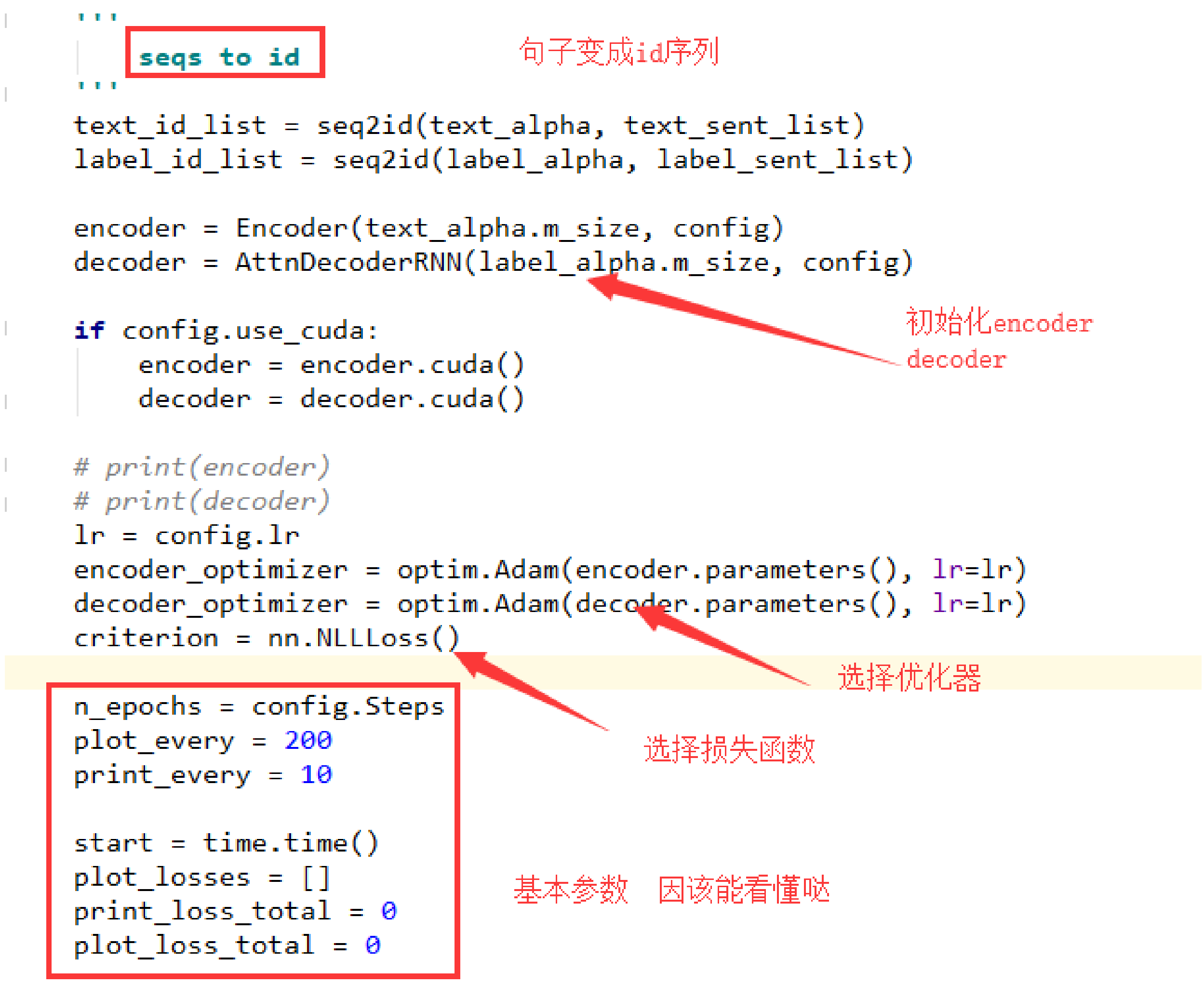


```
'''
    create dictionary
'''
# print(text_sent_list)
text_word_state = {'SOS': 10, 'EOS': 10, PAD: 10}
label_word_state = {'SOS': 10, 'EOS': 10, PAD: 10}
for line in text_sent_list:
    for word in line:
        if word not in text_word_state:
            text_word_state[word] = 1
        else:
            text_word_state[word] += 1
for line in label_sent_list:
    for word in line:
        if word not in label_word_state:
            label_word_state[word] = 1
        else:
            label_word_state[word] += 1
```

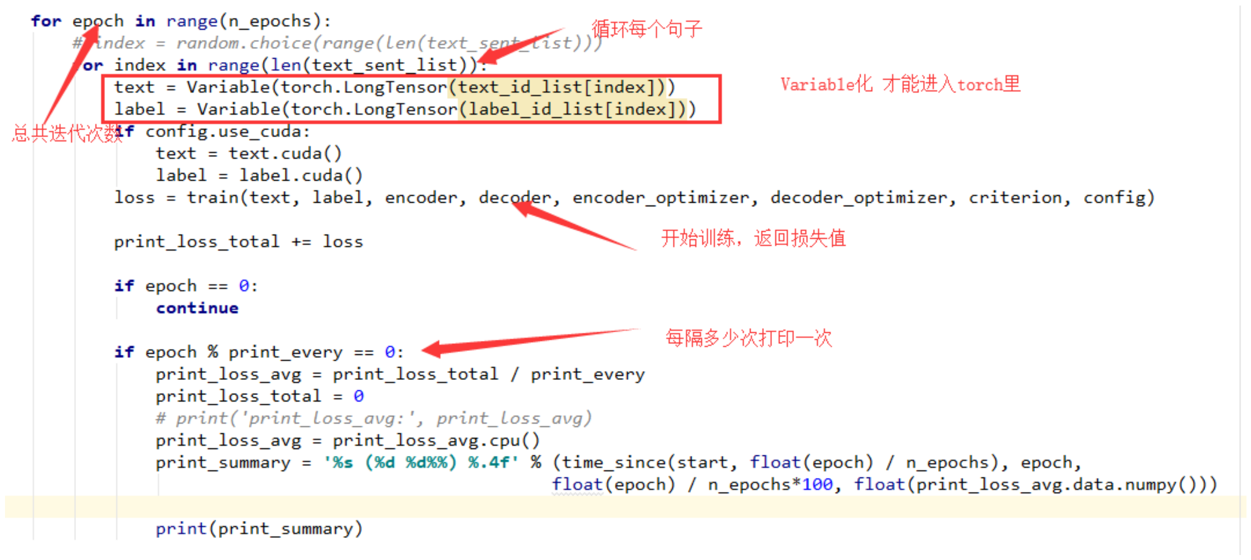
放到Alphabet类里



```python
'''
    create Alphabet
'''
text_alpha = Alphabet()
label_alpha = Alphabet()
text_alpha.initial(text_word_state)
label_alpha.initial(label_word_state)

# print(text_alpha.id2string)
print('text word size:', text_alpha.m_size)
print('label word size:', label_alpha.m_size)
# print(label_alpha.id2string)
```



```python
'''
    seqs to id
'''
text_id_list = seq2id(text_alpha, text_sent_list)
label_id_list = seq2id(label_alpha, label_sent_list)

encoder = Encoder(text_alpha.m_size, config)
decoder = AttnDecoderRNN(label_alpha.m_size, config)

if config.use_cuda:
    encoder = encoder.cuda()
    decoder = decoder.cuda()

# print(encoder)
# print(decoder)
lr = config.lr
encoder_optimizer = optim.Adam(encoder.parameters(), lr=lr)
decoder_optimizer = optim.Adam(decoder.parameters(), lr=lr)
criterion = nn.NLLLoss()

n_epochs = config.Steps
plot_every = 200
print_every = 10

start = time.time()
plot_losses = []
print_loss_total = 0
plot_loss_total = 0
```

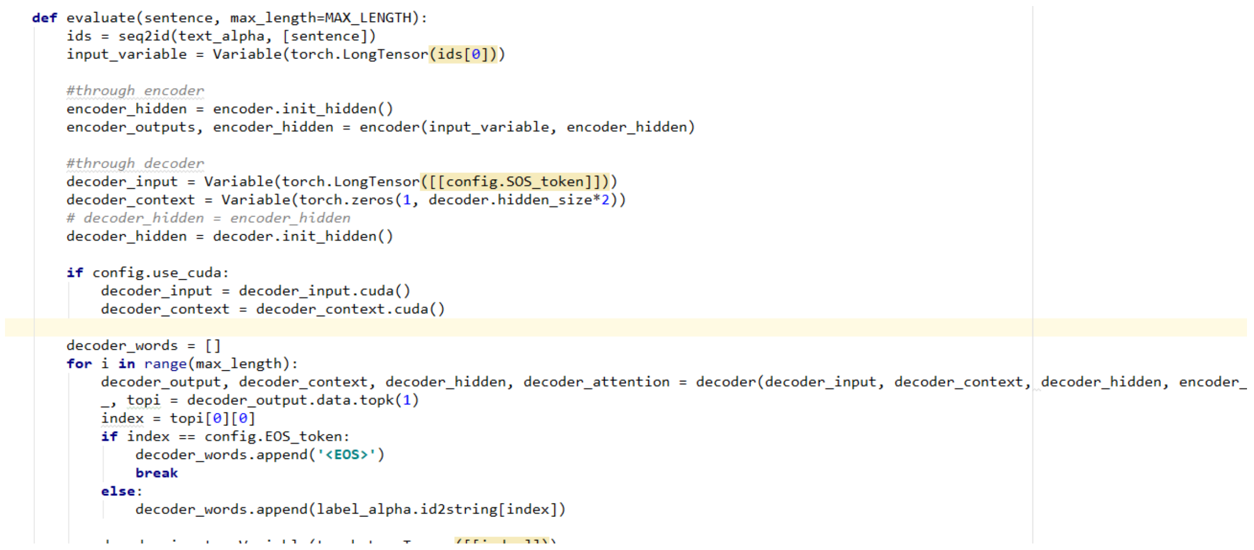


```
for epoch in range(n_epochs):
    # index = random.choice(range(len(text_sent_list)))
    for index in range(len(text_sent_list)):
        text = Variable(torch.LongTensor(text_id_list[index]))
        label = Variable(torch.LongTensor(label_id_list[index]))
        if config.use_cuda:
            text = text.cuda()
            label = label.cuda()
        loss = train(text, label, encoder, decoder, encoder_optimizer, decoder_optimizer, criterion, config)

        print_loss_total += loss

        if epoch == 0:
            continue

        if epoch % print_every == 0:
            print_loss_avg = print_loss_total / print_every
            print_loss_total = 0
            # print('print_loss_avg:', print_loss_avg)
            print_loss_avg = print_loss_avg.cpu()
            print_summary = '%s (%d %d%%) %.4f' % (time_since(start, float(epoch) / n_epochs), epoch,
                                                   float(epoch) / n_epochs*100, float(print_loss_avg.data.numpy()))

            print(print_summary)
```

每次大循环完 保存模型，接着再载入模型，进行评估

```
fmodel = 'seq2seq-%d.param' % n_epochs
torch.save([encoder, decoder], fmodel)
encoder, decoder = torch.load(fmodel)
```

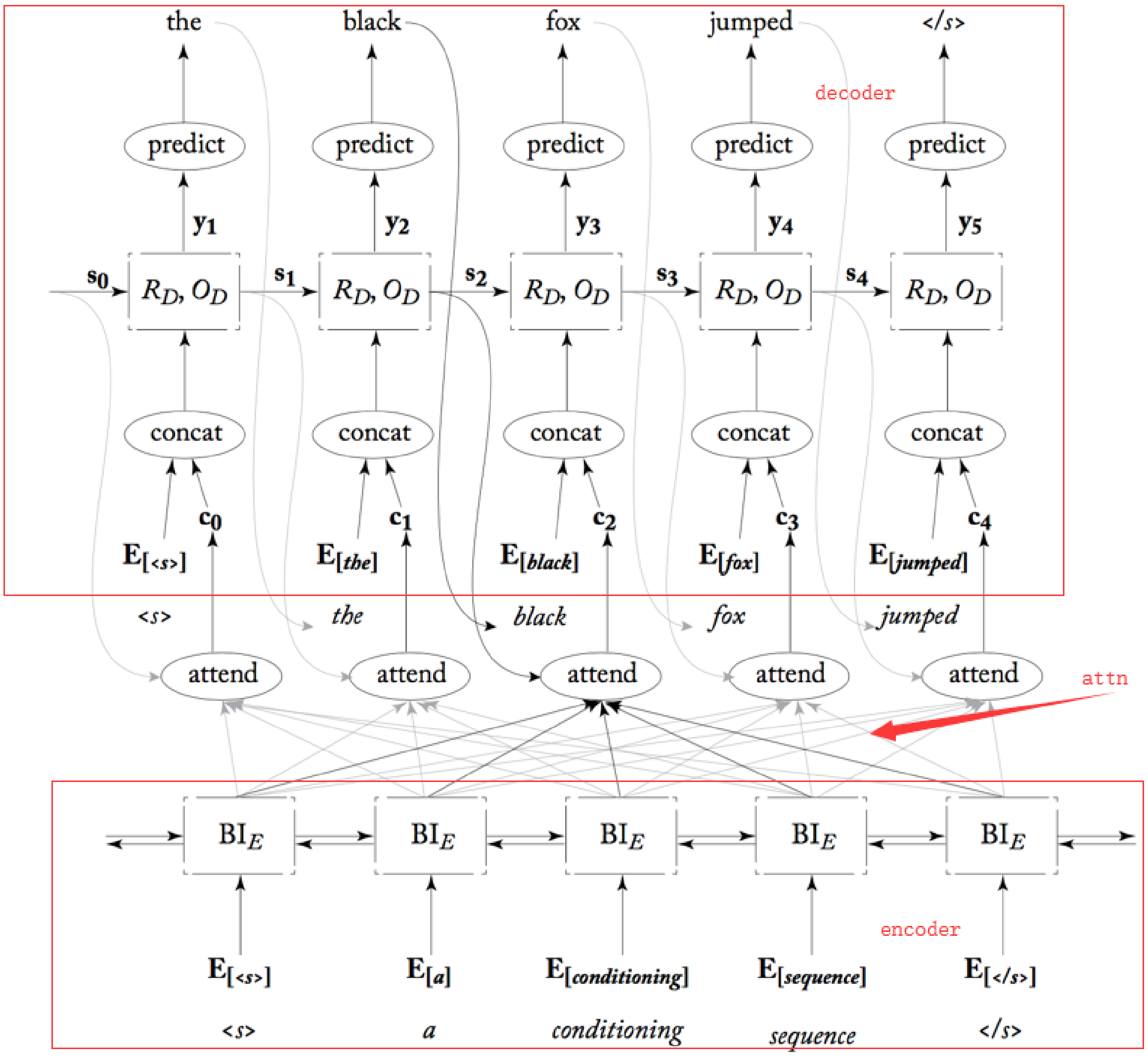
评估的过程和训练的过程差不多 但是没有更新参数的过程。所以 直接讲Train的过程、



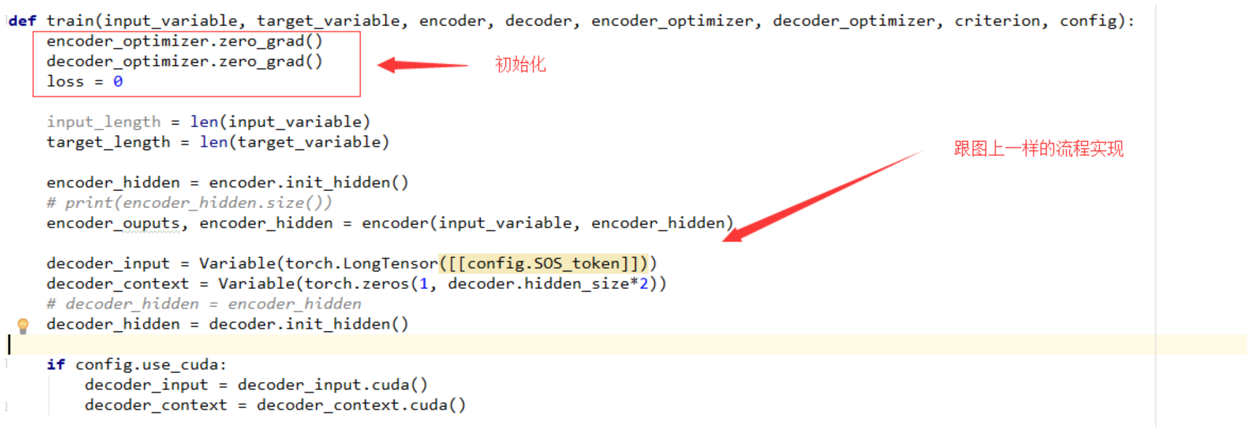
```
def evaluate(sentence, max_length=MAX_LENGTH):
    ids = seq2id(text_alpha, [sentence])
    input_variable = Variable(torch.LongTensor(ids[0]))

    #through encoder
    encoder_hidden = encoder.init_hidden()
    encoder_outputs, encoder_hidden = encoder(input_variable, encoder_hidden)

    #through decoder
    decoder_input = Variable(torch.LongTensor([[config.SOS_token]]))
    decoder_context = Variable(torch.zeros(1, decoder.hidden_size*2))
    # decoder_hidden = encoder_hidden
    decoder_hidden = decoder.init_hidden()

    if config.use_cuda:
        decoder_input = decoder_input.cuda()
        decoder_context = decoder_context.cuda()

    decoder_words = []
    for i in range(max_length):
        decoder_output, decoder_context, decoder_hidden, decoder_attention = decoder(decoder_input, decoder_context, decoder_hidden, encoder_
        _, topi = decoder_output.data.topk(1)
        index = topi[0][0]
        if index == config.EOS_token:
            decoder_words.append('<EOS>')
            break
        else:
            decoder_words.append(label_alpha.id2string[index])
```

下面开始说train：

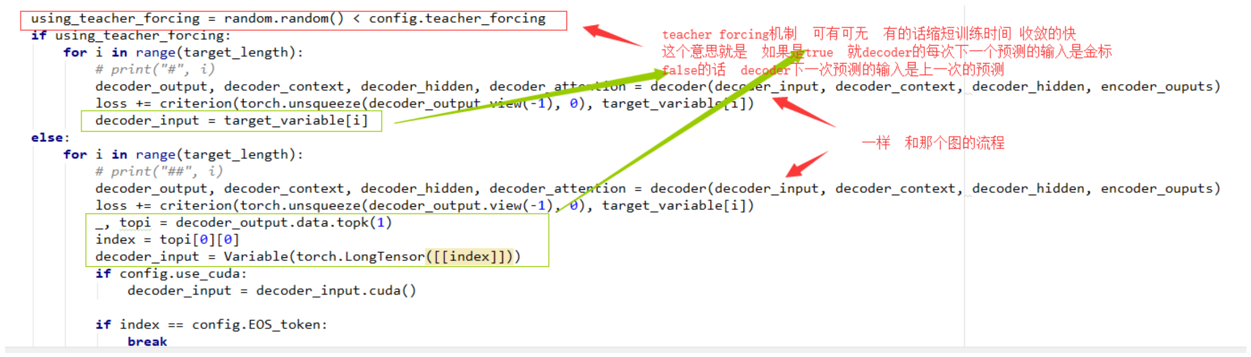
先说模型，你之前给我看的图片流程，和这个其实基本差不多的。所以，那个能懂，这个也能懂。



接下来看train流程：



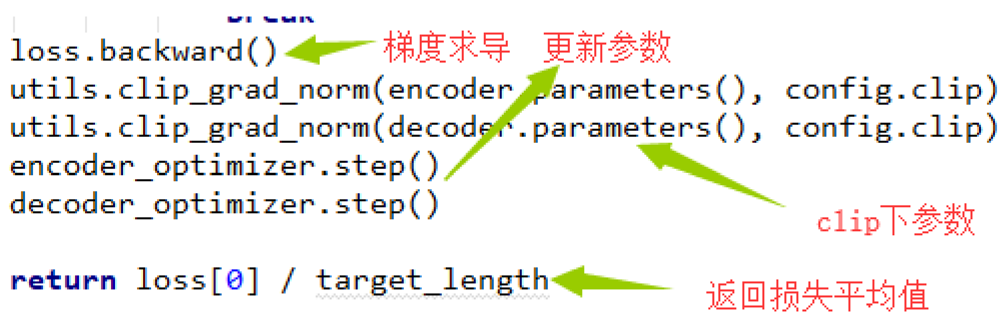
```python
def train(input_variable, target_variable, encoder, decoder, encoder_optimizer, decoder_optimizer, criterion, config):
    encoder_optimizer.zero_grad()
    decoder_optimizer.zero_grad()
    loss = 0

    input_length = len(input_variable)
    target_length = len(target_variable)

    encoder_hidden = encoder.init_hidden()
    # print(encoder_hidden.size())
    encoder_ouputs, encoder_hidden = encoder(input_variable, encoder_hidden)

    decoder_input = Variable(torch.LongTensor([[config.SOS_token]]))
    decoder_context = Variable(torch.zeros(1, decoder.hidden_size*2))
    # decoder_hidden = encoder_hidden
    decoder_hidden = decoder.init_hidden()

    if config.use_cuda:
        decoder_input = decoder_input.cuda()
        decoder_context = decoder_context.cuda()
```



```python
using_teacher_forcing = random.random() < config.teacher_forcing
if using_teacher_forcing:
    for i in range(target_length):
        # print("#", i)
        decoder_output, decoder_context, decoder_hidden, decoder_attention = decoder(decoder_input, decoder_context, decoder_hidden, encoder_ouputs)
        loss += criterion(torch.unsqueeze(decoder_output.view(-1), 0), target_variable[i])
        decoder_input = target_variable[i]
else:
    for i in range(target_length):
        # print("##", i)
        decoder_output, decoder_context, decoder_hidden, decoder_attention = decoder(decoder_input, decoder_context, decoder_hidden, encoder_ouputs)
        loss += criterion(torch.unsqueeze(decoder_output.view(-1), 0), target_variable[i])
        _, topi = decoder_output.data.topk(1)
        index = topi[0][0]
        decoder_input = Variable(torch.LongTensor([[index]]))
        if config.use_cuda:
            decoder_input = decoder_input.cuda()

        if index == config.EOS_token:
            break
```

teacher forcing机制　可有可无　有的话缩短训练时间 收敛的快
这个意思就是　如果是true　就decoder的每次下一个预测的输入是金标
false的话　decoder下一次预测的输入是上一次的预测

一样　和那个图的流程



```python
loss.backward()
utils.clip_grad_norm(encoder.parameters(), config.clip)
utils.clip_grad_norm(decoder.parameters(), config.clip)
encoder_optimizer.step()
decoder_optimizer.step()

return loss[0] / target_length
```

梯度求导　更新参数

clip下参数

返回损失平均值

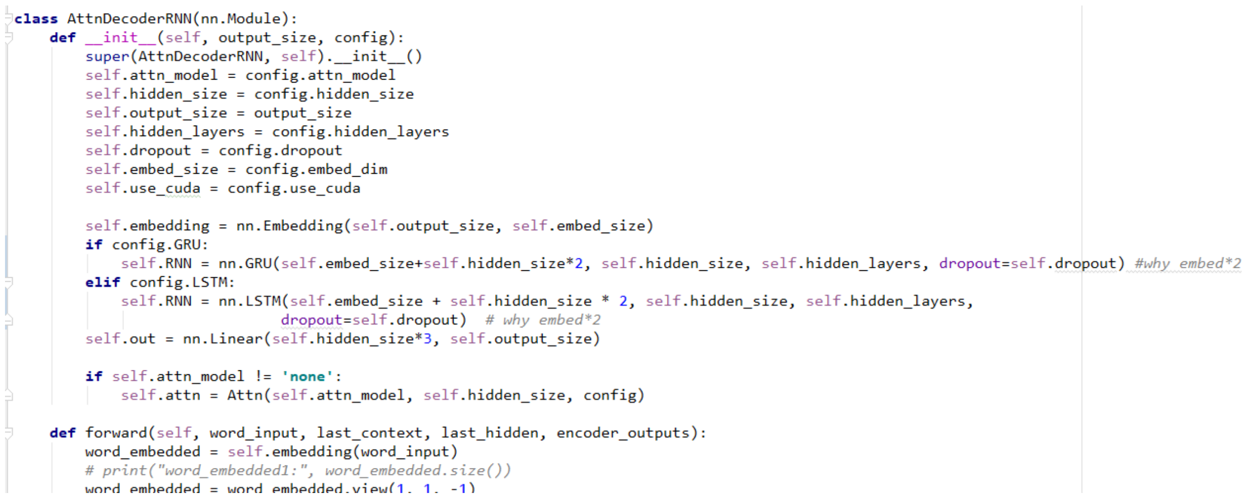
attn中的评价得分

```python
def score(self, hidden, encoder_output):
    if self.method == 'dot':
        energy = hidden.dot(encoder_output)
        return energy

    elif self.method == 'general':
        energy = self.attn(encoder_output)
        energy = energy.squeeze(0)
        hidden = hidden.squeeze(0)
        # print(energy.size())
        # print(hidden.size())
        energy = hidden.dot(energy)
        # print(energy.size())
        return energy

    elif self.method == 'concat':
        energy = self.attn(torch.cat((hidden, encoder_output), 1))
        energy = self.other.dot(energy)
        return energy
```

根据的公式是：

$$score(h_t, \bar{h}_s) = \begin{cases} h_t^\top \bar{h}_s & dot \\ h_t^\top \mathbf{W}_a \bar{h}_s & general \\ v_a^\top \mathbf{W}_a [h_t; \bar{h}_s] & concat \end{cases}$$

decoder


```
class AttnDecoderRNN(nn.Module):
    def __init__(self, output_size, config):
        super(AttnDecoderRNN, self).__init__()
        self.attn_model = config.attn_model
        self.hidden_size = config.hidden_size
        self.output_size = output_size
        self.hidden_layers = config.hidden_layers
        self.dropout = config.dropout
        self.embed_size = config.embed_dim
        self.use_cuda = config.use_cuda

        self.embedding = nn.Embedding(self.output_size, self.embed_size)
        if config.GRU:
            self.RNN = nn.GRU(self.embed_size+self.hidden_size*2, self.hidden_size, self.hidden_layers, dropout=self.dropout) #why embed*2
        elif config.LSTM:
            self.RNN = nn.LSTM(self.embed_size + self.hidden_size * 2, self.hidden_size, self.hidden_layers,
                                  dropout=self.dropout)  # why embed*2
        self.out = nn.Linear(self.hidden_size*3, self.output_size)

        if self.attn_model != 'none':
            self.attn = Attn(self.attn_model, self.hidden_size, config)

    def forward(self, word_input, last_context, last_hidden, encoder_outputs):
        word_embedded = self.embedding(word_input)
        # print("word_embedded1:", word_embedded.size())
        word_embedded = word_embedded.view(1, 1, -1)
```

其实也就是图中的流程。你如果想了解具体的，得慢慢熟悉pytorch啊。

如果只是跑数据 那就没啥

你先运行起来。有问题再问我