机器翻译

1 任务

用神经网络方法实现 中-->英 翻译系统。

1.1 模型要求：

（1）RNN + Attention 模型

（2）Transformer 模型

任选其中一种模型。

1.2 数据及平台：

(1）数据集：/data，6834 个中英平行语对。/cn.txt 和/en.txt 的每一行一一对应

(2）模型实现可用平台 Tensorflow 或 Pytorch

1.3 提交内容：

(1）整个工程项目：数据处理代码，模型核心代码，训练得到的模型文件

(2）项目说明文件：模型参数设置，模型测试详细过程

Note：本次作业实现了以上提到的两种模型，并对模型加以比较。

2 Transformer

2.1 数据处理

数据处理部分主要完成以下几个部分：数据读取、构造字典、构造词向量、构造 batch 等。

数据读取部分：读取文件，去除一些罕见的符号（只保留[，。？ ！,.?!]以及数字），然后按照设置的数据集划分比例划分出训练集和测试集。

```
load_data():
ch_sentences =[re.sub(u"[^0-9\u4e00-\u9fa5。，？！]+",' ',line)
               for line in codecs.open(Config.source_data_file, 'r', 'utf-8').readlines()]
en_sentences =[re.sub(u"[.,?!]+", lambda x:' '+x.group(0),re.sub(u"[^0-9a-zA-Z.,?!']+", ' ', line
               for line in codecs.open(Config.target_data_file, 'r', 'utf-8').readlines()]
split_index = int(len(ch_sentences) * Config.split)
train_ch_sents = ch_sentences[:split_index]
train_en_sents = en_sentences[:split_index]
test_ch_sents = ch_sentences[split_index:]
test_en_sents = en_sentences[split_index:]
return train_ch_sents, train_en_sents, test_ch_sents, test_en_sents
```

字典构造：只对词频数>阈值的词 设置字典。然后在词典中插入以下四个符

号。

```
PAD = '<pad>'
UNK = '<unk>'
BOS = '<s>'
EOS = '</s>'
```

```
def build_dictionary(sentences):
    dic = collections.OrderedDict()
    for sentence in sentences:
        for word in sentence.strip().split():
            word = word.strip()
            if word=='' or word =='\u3000':
                continue
            if word not in dic:
                dic[word] = 1
            else:
                dic[word] += 1
    vocab = [word for word in dic if word!='' and word !='\u3000' and dic[word] >= Config.min_count]
    vocab.insert(0, PAD)
    vocab.insert(1, UNK)
    vocab.insert(2, BOS)
    vocab.insert(3, EOS)
    word2idx = {word: idx for idx, word in enumerate(vocab)}
    idx2word = {idx: word for idx, word in enumerate(vocab)}
    #print(word2idx)
    return word2idx, idx2word
```

词向量构造：对每个词做映射，获得向量表示，此处只选择那些句子长度小于设定的长度阈值的数据集作为最后的训练集或测试集。

```
def word_mapping(ch_sents, en_sents, ch_word2idx, en_word2idx):
    ch_idxes, en_idxes, ches, enes = [], [], [], []
    for ch_sent, en_sent in zip(ch_sents, en_sents):
        ch_idx = [ch_word2idx.get(word, ch_word2idx[UNK])
                  for word in (ch_sent.strip() + ' ' + EOS).split() if word!='' and word !='\u3000']
        en_idx = [en_word2idx.get(word, en_word2idx[UNK])
                  for word in (en_sent.strip() + ' ' + EOS).split() if word!='' and word !='\u3000']
        if max(len(ch_idx), len(en_idx)) <= Config.max_length:
            ch_idxes.append(ch_idx)
            en_idxes.append(en_idx)
            ches.append(ch_sent)
            enes.append(en_sent)
    return ch_idxes, en_idxes, ches, enes
```

获取数据：这部分是与模型直接交互，用于获取最后处理好的训练集源语句、词向量表示、字典大小等。

```
def preprocess_data(isTrain=True):
    train_ch_sents, train_en_sents, test_ch_sents, test_en_sents = load_data()
    ch_word2idx, ch_idx2word = build_dictionary(train_ch_sents+test_ch_sents)
    en_word2idx, en_idx2word = build_dictionary(train_en_sents+test_en_sents)
    len_ch_vocab = len(ch_word2idx)
    len_en_vocab = len(en_word2idx)
    if isTrain:
        ch_idxes, en_idxes, ches, enes = word_mapping(train_ch_sents, train_en_sents,
                                                      ch_word2idx, en_word2idx)
    else:
        ch_idxes, en_idxes, ches, enes = word_mapping(test_ch_sents, test_en_sents,
                                                      ch_word2idx, en_word2idx)
    # padding
    ch_idxes = [seq_idx + (Config.max_length - len(seq_idx)) * [ch_word2idx[PAD]]
                for seq_idx in ch_idxes]
    en_idxes = [seq_idx + (Config.max_length - len(seq_idx)) * [en_word2idx[PAD]]
                for seq_idx in en_idxes]
    if isTrain:
        return ch_idxes, en_idxes, ches, enes, len_ch_vocab, len_en_vocab
    else:
        return ch_idxes, en_idx2word, ches, enes, len_ch_vocab, len_en_vocab
```

获取 batch：使用 TensorFlow 提供的工具来获取一个 batch queue。

```
def get_batch_data():
    ch_idxes, en_idxes, ches, enes, len_ch_vocab, len_en_vocab = preprocess_data(True)
    num_batch = len(ch_idxes)//Config.batch_size
    ch_idxes = np.array(ch_idxes)
    en_idxes = np.array(en_idxes)
    ch_idxes = tf.convert_to_tensor(ch_idxes, tf.int32)
    en_idxes = tf.convert_to_tensor(en_idxes, tf.int32)
    print(ch_idxes.get_shape())
    input_queues = tf.train.slice_input_producer([ch_idxes, en_idxes])
    ch, en = tf.train.shuffle_batch(input_queues, num_threads=8, batch_size=Config.bat
    return ch, en, num_batch, len_ch_vocab, len_en_vocab
```

2.2 模型核心部分

2.2.1 总体架构

该模型架构如下所示，代码中采用模块化设计。

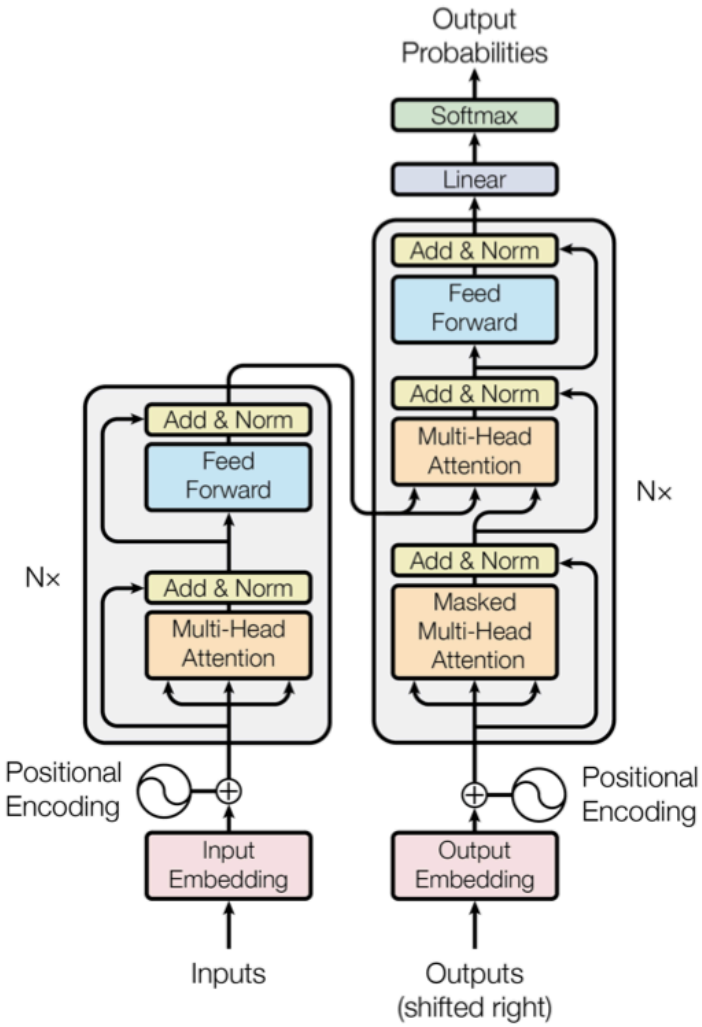
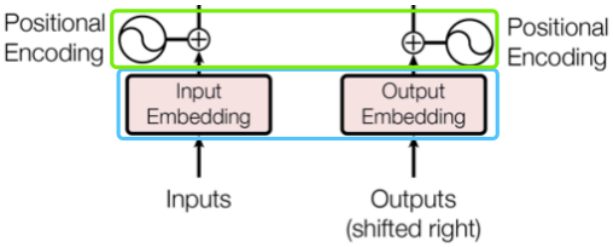


Figure 1: The Transformer - model architecture.

```
with self.graph.as_default():
    self.pre_data(istraining) # 设置placeholder，设置模型输入
    self.encoder(istraining)  # encoder层
    self.decoder(istraining)  # decoder层
    self.pred(istraining)     # 训练
```

2.2.1 词向量输入层：



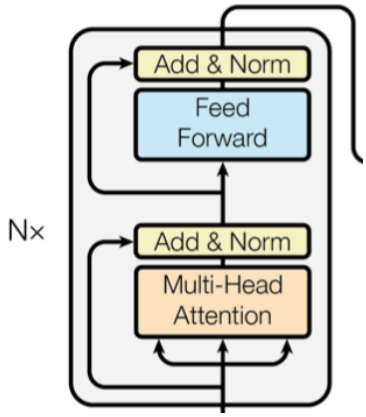
主要处理 encoder 和 decoder 的输入词向量、位置词向量。位置词向量可以采用固定的计算方法，如下所示；也可以直接采用随机生成词向量，一同参与训练。

$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

```
# 词向量编码
def EmbeddingsWithSoftmax(inputs, vocab_size, zero_pad=True, scope='embedding'):
    with tf.variable_scope(scope):
        lookup_table = tf.get_variable('lookup_table',
                                       shape=[vocab_size, Config.dim_model],
                                       dtype=tf.float32,
                                       initializer=tf.contrib.layers.xavier_initializer())
        if zero_pad:
            lookup_table = tf.concat((tf.zeros(shape=[1, Config.dim_model]), lookup_table[1:, :]), 0)
            embedding = tf.nn.embedding_lookup(lookup_table, inputs)
            embedding = embedding * (Config.dim_model ** 0.5)
        else:
            embedding = tf.nn.embedding_lookup(lookup_table, inputs)
        return embedding
```

```
# 位置编码
# PE(pos, 2i) = sin(pos / 10000^{2i/d_model})
# PE(pos, 2i+1) = cos(pos / 10000^{2i/d_model})
def PositionalEncoding(inputs, scope='pos_embedding'):
    len_batch, len_sent = inputs.get_shape().as_list()
    with tf.variable_scope(scope):
        pos_ind = tf.tile(tf.expand_dims(tf.range(tf.shape(inputs)[1]), 0), [tf.shape(inputs)[0], 1])
        pos_enc = np.array(
            [[pos / np.power(10000., 2. * (i) / Config.dim_model)
              for i in range(Config.dim_model)]
             for pos in range(len_sent)
             ])
        pos_enc[:, 0::2] = np.sin(pos_enc[:, 0::2])  # dim 2i
        pos_enc[:, 1::2] = np.cos(pos_enc[:, 1::2])  # dim 2i+1
        lookup_table = tf.convert_to_tensor(pos_enc, dtype=tf.float32)
        embedding = tf.nn.embedding_lookup(lookup_table, pos_ind)
        return embedding
```

2.2.2 encoder 层

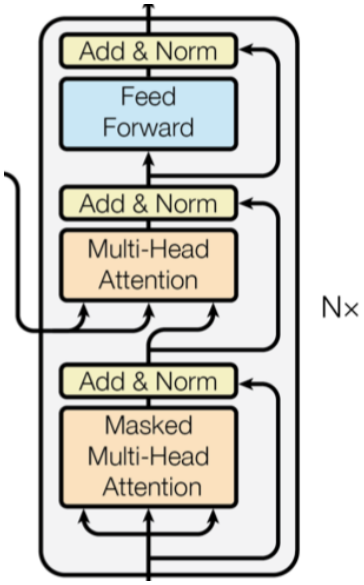


原作者采用 6 个完全相同设计的单个 encoder block 形成的 stack 来组成一个完整的 encoder，单层 encoder 都由两个子层组成：multi-head self-attention 机制和 positionwise 全连接前馈网络组成。两个子层最后都运用了 residual connection 并做了 layer normalization。

```
def EncoderLayer(inputs_embedding, istraining=True):
    multi = MultiHeadedAttention(inputs_embedding, inputs_embedding, inputs_embedding, istraining, F
    multi_layernorm = LayerNorm(inputs_embedding, multi)
    fft = PositionwiseFeedForward(multi_layernorm)
    fft_layernorm = LayerNorm(multi_layernorm, fft)
    return fft_layernorm


def Encoder(inputs_embedding, istraining=True):
    encoder_outputs = inputs_embedding
    for i in range(Config.num_block):
        with tf.variable_scope("encoder_num_blocks_{}".format(i)):
            encoder_outputs = EncoderLayer(encoder_outputs, istraining)
    return encoder_outputs
```

2.2.3 decoder 层



decoder 子层和 encoder 子层设计基本相似，只是多了一个 masked multi-head self-attention。这主要是为了保证当前预测所依赖的知识只是它已经预测出的知

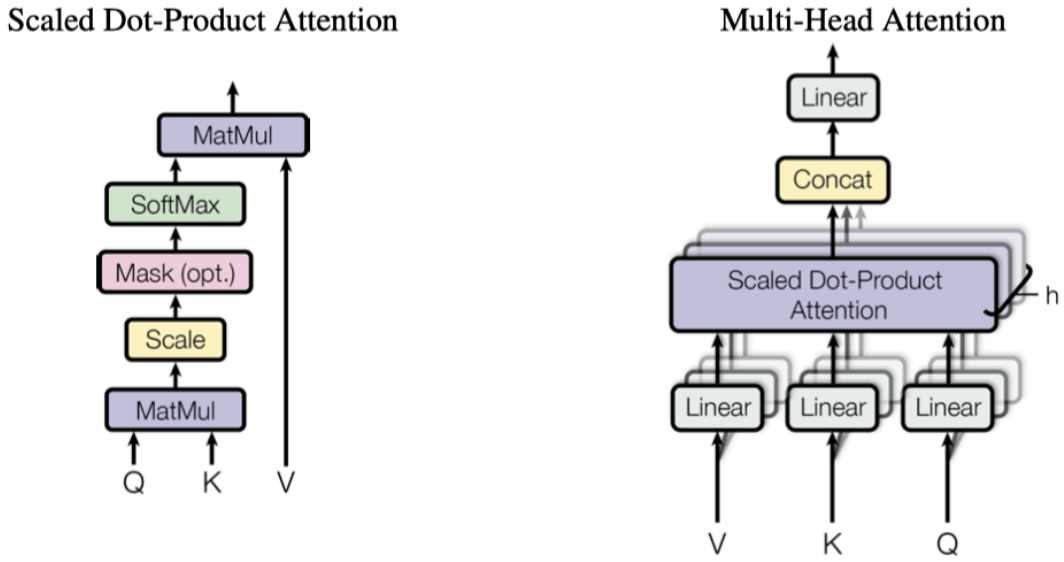
识，对句子后面的知识目前是不清楚的。此外，此处的输入来自两部分：自身的输出值以及 encoder 层的输出值。

```
def Encoder(inputs_embedding, istraining=True):
    encoder_outputs = inputs_embedding
    for i in range(Config.num_block):
        with tf.variable_scope("encoder_num_blocks_{}".format(i)):
            encoder_outputs = EncoderLayer(encoder_outputs, istraining)
    return encoder_outputs


def DecoderLayer(inputs_embedding, encoder_outputs, istraining=True):
    masked_multi = MultiHeadedAttention(inputs_embedding, inputs_embedding, inputs_embedding, istrair
    masked_multi_layernorm = LayerNorm(inputs_embedding, masked_multi)
    multi = MultiHeadedAttention(masked_multi_layernorm, encoder_outputs, encoder_outputs, istraining
    multi_layernorm = LayerNorm(masked_multi_layernorm, multi)
    fft = PositionwiseFeedForward(multi_layernorm)
    fft_layernorm = LayerNorm(multi_layernorm, fft)
    return fft_layernorm
```

2.2.4 Multi-Head Attention 子层

这部分是精髓所在，主要架构如下所示：



采用 Multi-Head Attention，分成多个 header 分别计算，最后训练处多个具有不同倚重的特征，能有更好的性能表现。最后将多个 header 进行 concat。

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, ..., \text{head}_\text{h})W^O$$
$$\text{where head}_\text{i} = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)$$

```
def MultiHeadedAttention(query, key, value, istraining=True, mask=True):
    Q = tf.layers.dense(query, Config.dim_model, activation=tf.nn.relu)
    K = tf.layers.dense(key,   Config.dim_model, activation=tf.nn.relu)
    V = tf.layers.dense(value, Config.dim_model, activation=tf.nn.relu)
    # 切分
    Q = tf.concat(tf.split(Q, Config.num_header, axis=2), axis=0)
    K = tf.concat(tf.split(K, Config.num_header, axis=2), axis=0)
    V = tf.concat(tf.split(V, Config.num_header, axis=2), axis=0)
    outputs = Attention(Q, K, V, mask, istraining)
    outputs = tf.concat(tf.split(outputs, Config.num_header, axis=0), axis=2)
    return outputs
```

每个小的 attention 采用 Scaled Dot-Product Attention 设计。计算方式如下：

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

```
def Attention(Q, K, V, mask=False, istraining=True):
    scores = tf.matmul(Q, tf.transpose(K, [0, 2, 1]))
    scores = scores / (K.get_shape().as_list()[-1] ** 0.5)
    if mask:
        diag_vals = tf.ones_like(scores[0, :, :])
        tril = tf.linalg.LinearOperatorLowerTriangular(diag_vals).to_dense()
        masks = tf.tile(tf.expand_dims(tril, 0), [tf.shape(scores)[0], 1, 1])
        paddings = tf.ones_like(masks) * (-2 ** 32 + 1)
        scores = tf.where(tf.equal(masks, 0), paddings, scores)
    p_attn = tf.nn.softmax(scores)
    p_attn = tf.layers.dropout(p_attn, rate=Config.dropout_rate,
                               training=tf.convert_to_tensor(istraining))
    return tf.matmul(p_attn, V)
```

2.2.5 layer normalization 层

此处使用残差网络，然后使用 layer normalization。

```
def LayerNorm(x, sublayer):
    x = tf.add(x, sublayer)
    inputs = x
    epsilon = 1e-8
    inputs_shape = inputs.get_shape()
    params_shape = inputs_shape[-1:]
    mean, variance = tf.nn.moments(inputs, [-1], keep_dims=True)
    beta = tf.Variable(tf.zeros(params_shape))
    gamma = tf.Variable(tf.ones(params_shape))
    normalized = (inputs - mean) / ((variance + epsilon) ** (.5))
    outputs = gamma * normalized + beta
    # outputs =  tf.contrib.layers.layer_norm(x)
    return outputs
```

2.2.6 Position-wise 前馈网络

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

```
def PositionwiseFeedForward(inputs):
    # FFN(x) = max(0, x*W1+b1)*W2 + b2
    outputs = tf.layers.conv1d(inputs, filters=4 * Config.dim_model, kernel_size=1, activation=tf.nn.relu)
    outputs = tf.layers.conv1d(outputs, filters=Config.dim_model, kernel_size=1)
    return outputs
```

2.2.7 训练

使用 Adam 优化器，具体参数也使用原论文中的设置。

```
def pred(self, istraining=True):
    self.logits = tf.layers.dense(self.decoder_outputs, self.en_vocab_size)
    self.preds = tf.to_int32(tf.argmax(self.logits, dimension=-1))
    self.istarget = tf.to_float(tf.not_equal(self.en, 0))
    self.acc = tf.reduce_sum(tf.to_float(
        tf.equal(self.preds, self.en)) * self.istarget) / (tf.reduce_sum(self.istarget))
    tf.summary.scalar('acc', self.acc)
    # 训练
    if istraining:
        self.en_smoothed = LabelSmoothing(tf.one_hot(self.en, depth=self.en_vocab_size))
        self.loss = tf.nn.softmax_cross_entropy_with_logits_v2(logits=self.logits, labels=self.en_smoothed)
        self.mean_loss = tf.reduce_sum(self.loss * self.istarget) / (tf.reduce_sum(self.istarget))
        self.global_step = tf.Variable(0, name='global_step', trainable=False)
        self.optimizer = tf.train.AdamOptimizer(learning_rate=Config.lr, beta1=0.9, beta2=0.98, epsilon=1e-8)
        self.train_op = self.optimizer.minimize(self.mean_loss, global_step=self.global_step)
        tf.summary.scalar('mean_loss', self.mean_loss)
        self.merged = tf.summary.merge_all()
```

训练时使用 Label Smoothing，参数为 0.1。

```
def LabelSmoothing(inputs, epsilon=0.1):
    K = inputs.get_shape().as_list()[-1]
    return ((1 - epsilon) * inputs) + (epsilon / K)
```

2.3 模型参数设置

模型参数设置如下：

```
class Config:

    source_data_file = './data/cn.txt'    # 翻译源语言文件
    target_data_file = './data/en.txt'    # 翻译目标语言文件
    save_models_dir = './models/'         # 模型存储目录
    evaluate_result_dir = './result/'     # 评估结果目录
    split = 0.95                # 数据集划分比例
    batch_size = 32             #
    lr = 0.0001                 #
    lr_decay = 0.98             #
    dim_model = 512             # 单层模型单元数目
    num_block = 6               # encoder和decoder的block数目
    num_header = 8              # multiheader的header数目
    num_epoch = 25              # 训练轮数
    min_count = 2               # 词频最小阈值
    max_length = 32             # 语句最大长度
    dropout_rate = 0.2          #
```

2.4 模型测试过程

模型测试分为两部分：

第一步：训练模型

python3 main.py train

第二步：测试模型

python3 main.py test

下图为模型训练过程保存的模型：

```
-rw-r--r-- 1 root root  24K Jan  5 17:02 model_epoch_19.index
-rw-r--r-- 1 root root 6.1M Jan  5 17:02 model_epoch_19.meta
-rw-r--r-- 1 root root 641M Jan  5 17:25 model_epoch_20.data-00000-of-00001
-rw-r--r-- 1 root root  24K Jan  5 17:25 model_epoch_20.index
-rw-r--r-- 1 root root 6.1M Jan  5 17:25 model_epoch_20.meta
-rw-r--r-- 1 root root 641M Jan  5 17:54 model_epoch_21.data-00000-of-00001
-rw-r--r-- 1 root root  24K Jan  5 17:54 model_epoch_21.index
-rw-r--r-- 1 root root 6.1M Jan  5 17:54 model_epoch_21.meta
-rw-r--r-- 1 root root 641M Jan  5 18:07 model_epoch_22.data-00000-of-00001
-rw-r--r-- 1 root root  24K Jan  5 18:07 model_epoch_22.index
-rw-r--r-- 1 root root 6.1M Jan  5 18:07 model_epoch_22.meta
-rw-r--r-- 1 root root 641M Jan  5 18:20 model_epoch_23.data-00000-of-00001
-rw-r--r-- 1 root root  24K Jan  5 18:20 model_epoch_23.index
-rw-r--r-- 1 root root 6.1M Jan  5 18:20 model_epoch_23.meta
(env_eric) [root@iZ25jr7xw0jZ 3nmt]#
```

下图为测试输出结果文件内容：

```
SOURCE>> 需要 建立 一个 联盟 ， 支持 国家 、 区域 和 全球 变革 。
TARGET>> A coalition of support for national, regional and global change is needed.
TRANSL>> There is a need to strengthen the global partnerships for the global economic and economic effo
rts.

SOURCE>> （ 以 英语 发言 ）
TARGET>> (spoke in English)
TRANSL>> (spoke in English)

SOURCE>> 在 这项 工作 中 ， 重要 的 是 提高 国家 的 渔业 和 海洋 管理 能力 。
TARGET>> In that effort, it is critical to develop country capacity for fisheries and oceans management.
TRANSL>> In this context, the challenge of building the creation of a growing capacity of the State and
small island

SOURCE>> 其 关键 是 建立 在 管理 的 生态系统 方针 基础 上 的 综合 海洋 规划 。
TARGET>> Its basis is integrated oceans planning, founded on an ecosystem approach to management.
TRANSL>> The strategy of the Republic of Tajikistan is the responsibility of the administering Powers to
 promote the framework <unk>
```

```
SOURCE>> 我国 的 基本 国情 与 诞生 马克思主义 的 西方 社会 与 诞生 列宁主义 的 俄国 的 情况 也 大 不 相同 。
TARGET>> china 's basic national conditions are also far different from the western society that gave birth to marxism and the russian circumst
ances in which leninism emerged  .
TRANSL>> china 's national conditions of the west , and the west <unk> from the west , and its important <unk> with other countries ' income di
stribution .

SOURCE>> 同时 坚持 理论 联系 实际 善于 应用 科学 理论 科学 知识 研究 和 解决 面临 的 实际 问题 。
TARGET>> at the same time  , they must adhere to integrating theory with practice and be good at studying and solving existing problems using s
cientific theory and knowledge  .
TRANSL>> at present , scientific and technological innovation are now developing many theoretical innovation and problems , to solve problems a
re required the use of our theoretical knowledge .

SOURCE>> 储蓄率 和 重 工业 比重 的 上升 是 一 个 国家 的 工业化 进入 到 中后期 阶段 的 标志 。
TARGET>> a rising savings rate and the growing importance of heavy industry are signs that a country 's industrialization has entered its middl
e and end stages  .
TRANSL>> the phenomenon of industrial and the construction of the west is a country 's most important <unk> is to be left for a country 's pric
e .
```

本次作业使用 nltk 工具包对翻译结果做评测，在测试集上最终的结果为：

```
Bleu Score = 5.168246637976368
```

可以说结果相当不理想，但在其它大语料短语句上测试时 bleu 值就比较理想。

2.4 问题与解决

问题 1：最初由于对模型的理解不足，decoder 的输入最初没有搞清楚，在编码的时候就出现了错误，即使使用大语料去训练仍然无法得到理想的结果，最后经过反复排查，才得知问题所在。

问题 2：当设置句子最大长度比较大时，由于语料比较小，而且句子长度基本都处于 20-32 之间，训练出来模型进行测试，测试结果很不理想，一直怀疑自己是否模型复现有误；最后设置一个比较小的句子长度阈值，在大语料中训练，结果还是比较理想的。从上面两张图也可以看出来，较长的语句翻译结果并不理想。

问题 3：由于代码中的字典构造是在程序运行时自己构造的，所以一旦改变了训练集，其字典也就发生了变化，这样训练好的模型就无法使用。解决方法是可以提前将字典保存好，在其它语料上仍然使用该字典，这样就可以测试以训练好的模型。

3 Seq2seq+Attention 模型

3.1 数据处理

数据处理部分和 transformer 实现有一些不同，此处需要对中文语料中的词分割成一个个字，然后存储在一个处理后的文件中，并在每段话前后加<s></s>标志。此外，吸取实现 transformer 中的教训，本次是将字典存储在文件中，这样即使中英文语料不同，也能使用该字典。

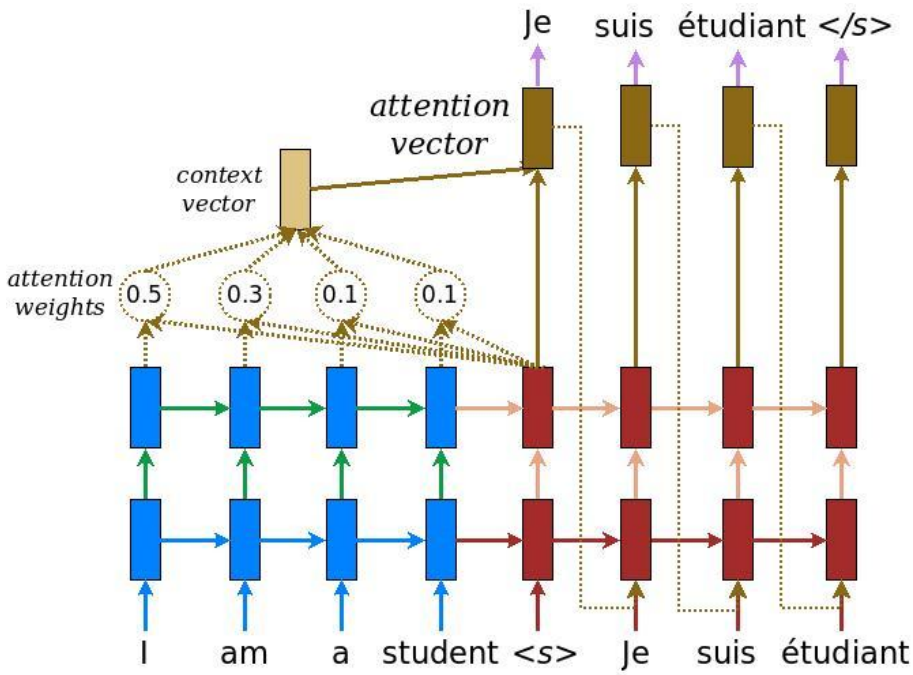
```
def clears(self,):
    data_en = 'data/en.txt'
    data_zh = 'data/cn.txt'
    data_en_clear = 'data/en_pro.txt'
    data_zh_clear = 'data/zh_pro.txt'
    with open(data_en, 'r',encoding='utf-8') as f_en:
        f_en_w = open(data_en_clear, 'w', encoding='utf-8')
        lineID = 0
        for line in f_en:
            if '<'==line[0] and '>' ==line[-2]:
                continue
            lineID += 1
            line = re.sub(u"[^0-9a-zA-Z.,?!']+",' ',line)
            line = re.sub(u"[.,?!]+", lambda x:' '+x.group(0),line)
            line = line.lower()
            f_en_w.write(line+'\n')
        print('english lines number:',lineID)
        f_en_w.close()

    with open(data_zh, 'r',encoding='utf-8') as f_zh:
        f_zh_w = open(data_zh_clear, 'w', encoding='utf-8')
        lineID = 0
        for line in f_zh:
            if '<' == line[0] and '>' == line[-2]:
                continue
            lineID += 1
            line = re.sub(u"[^0-9\u4e00-\u9fa5.,?!']+",'',line)  # 清除不需要的字符
            line = '<s> '+' '.join(line) +' </s>'
            f_zh_w.write(line + '\n')
        print('chinese lines number:',lineID)
        f_zh_w.close()
```

3.2 模型核心部分

3.2.1 整体框架

本次实现是在 Seq2seq 模型上加入 Attention 机制，总体框架如下图所示，encoder 和 decoder 都使用两层的 LSTM 实现。



3.2.2 encoder 层

使用两层 LSTM 实现。

```
with tf.variable_scope("encoder"):
    # 定义rnn网络
    def get_en_cell(hidden_dim):
        # 创建单个lstm
        enc_base_cell = tf.nn.rnn_cell.BasicLSTMCell(hidden_dim, forget_bias=1.0)
        return enc_base_cell
    self.enc_cell = tf.nn.rnn_cell.MultiRNNCell(
        [get_en_cell(self.config.hidden_dim) for _ in range(self.config.num_layers)])
    enc_output, self.enc_state= tf.nn.dynamic_rnn(cell=self.enc_cell,
                                                  inputs=embed_source_seqs,
                                                  sequence_length=self.source_length,
                                                  time_major=False,
                                                  dtype=tf.float32)
```

3.2.3 decoder 层

使用两层的 LSTM 实现。

```
with tf.variable_scope("decoder"):
    # 定义rnn网络
    def get_de_cell(hidden_dim):
        # 创建单个lstm
        dec_base_cell = tf.nn.rnn_cell.BasicLSTMCell(hidden_dim, forget_bias=1.0)
        return dec_base_cell
    self.dec_cell = tf.nn.rnn_cell.MultiRNNCell(
        [get_de_cell(self.config.hidden_dim) for _ in range(self.config.num_layers)])
    dec_output, self.dec_state= tf.nn.dynamic_rnn(self.dec_cell,
                                                  inputs=embed_target_seqs,
                                                  sequence_length=self.target_length,
                                                  initial_state=self.enc_state,  # 编
                                                  time_major=False,
                                                  dtype=tf.float32)
```

3.2.4 Attention 机制

本次实现使用的 Attention 机制是 BahdanauAttention，它是 Bahdanau 在论文 NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND

TRANSLATE 中提出的，整体 Attention 结构如下图：

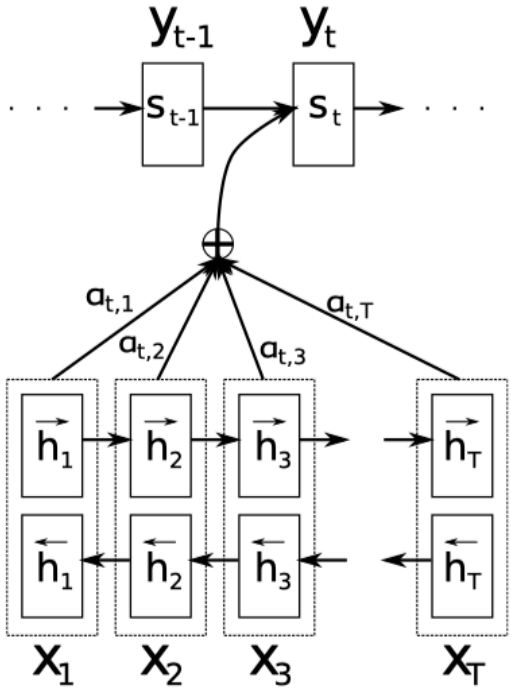


Figure 1: The graphical illustration of the proposed model trying to generate the t-th target word y_t given a source sentence $(x_1, x_2, \ldots, x_T)$.

```
with tf.variable_scope("attention"):
    # 选择注意力权重计算模型，BahdanauAttention是使用一个隐藏层的前馈网络，
    # memory_sequence_length是一个维度[batch_size]的张量，代表每个句子的长度
    attention_mechanism = tf.contrib.seq2seq.BahdanauAttention(
        self.config.hidden_dim, enc_output, memory_sequence_length=self.en_length)
    # 将解码器和注意力模型一起封装成更高级的循环网络
    attention_cell = tf.contrib.seq2seq.AttentionWrapper(
        self.dec_cell, attention_mechanism, attention_layer_size=self.config.hidden_dim)
    # 使用dynamic_rnn构造编码器，没有指定init_state,也就完全依赖注意力作为信息来源
    dec_output, self.dec_state = tf.nn.dynamic_rnn(
        attention_cell, inputs=embed_target_seqs, sequence_length=self.zh_length,
        time_major=False, dtype=tf.float32)
```

3.3 模型参数设置

模型参数设置如下：

```
class Config(object):
    source_data_file = './data/cn.txt'   # 翻译源语言文件
    target_data_file = './data/en.txt'   # 翻译目标语言文件
    save_models_dir = './models/'        # 模型存储目录
    save_model_path = 'seq2seq'          #保存模型文件
    evaluate_result_dir = './result/'    # 评估结果目录
    embedding_dim = 128                  # 词向量维度
    seq_length = 50                      # 序列长度
    target_vocab_size = 10000            # 词汇表达小
    source_vocab_size = 10000
    num_layers= 2                        # 隐藏层层数
    hidden_dim = 128                     # 隐藏层神经元
    share_emb_and_softmax = False
    train_keep_prob = 0.8                # dropout
    learning_rate = 1e-3                 # 学习率
    batch_size = 32                      #
    log_every_n = 20                     #
    save_every_n = 0                     #
    max_steps = 20000                    # 总迭代次数
```

3.4 模型测试过程

主要分为三部分

1）数据预处理

python3 data_util.py

处理后的语料如下：

```
<s> 目 前 粮 食 出 现 阶 段 性 过 剩 恰 好 可 以 以 粮 食 换 森 林 换 草 地 再 造 西 部 秀 美 山 川 。 </s>
<s> 工 程 的 立 项 设 计 都 要 充 分 论 证 反 复 比 较 使 工 程 经 得 起 时 间 检 验 。 </s>
<s> 建 设 现 代 化 的 气 象 水 文 通 信 设 施 加 强 雨 情 水 情 分 析 研 究 提 高 预 报 的 准 确 性 。 </s>
<s> 随 着 香 港 和 澳 门 的 回 归 祖 国 解 决 台 湾 问 题 突 出 地 摆 在 了 全 体 中 国 人 民 的 面 前 。 </s>
<s> 象 往 年 一 样 数 百 名 乡 亲 今 天 上 午 聚 集 在 钟 氏 祠 堂 前 的 广 场 举 行 团 拜 互 致 问 候 。 </s>
<s> 这 家 的 主 人 是 6 9 岁 的 退 休 公 务 员 钟 志 坚 他 有 三 子 一 女 及 3 名 孙 儿 。 </s>
<s> 林 村 乡 历 史 悠 久 居 民 的 先 祖 于 宋 朝 末 年 已 迁 徙 至 此 至 今 已 有 7 0 0 多 年 。 </s>
<s> 俄 代 总 统 普 京 日 前 在 俄 联 邦 安 全 会 议 上 明 确 指 出 新 军 事 学 说 是 对 北 约 的 回 答 。 </s>
<s> 为 了 得 到 别 人 的 尊 重 必 须 保 持 核 潜 力 将 核 武 器 作 为 遏 制 敌 人 的 政 治 工 具 。 </s>
<s> 军 事 专 家 指 出 俄 当 局 如 不 动 用 武 装 力 量 取 得 车 臣 战 争 的 胜 利 是 难 以 想 象 的 。 </s>
<s> 因 此 说 俄 武 装 力 量 只 用 来 对 付 外 来 侵 略 的 军 事 学 说 已 经 不 符 合 今 天 的 现 实 。 </s>
<s> 中 国 人 民 银 行 应 当 将 改 版 后 的 人 民 币 的 发 行 时 间 面 额 主 要 特 征 等 予 以 公 告 。 </s>
```

2）训练模型

python3 train.py

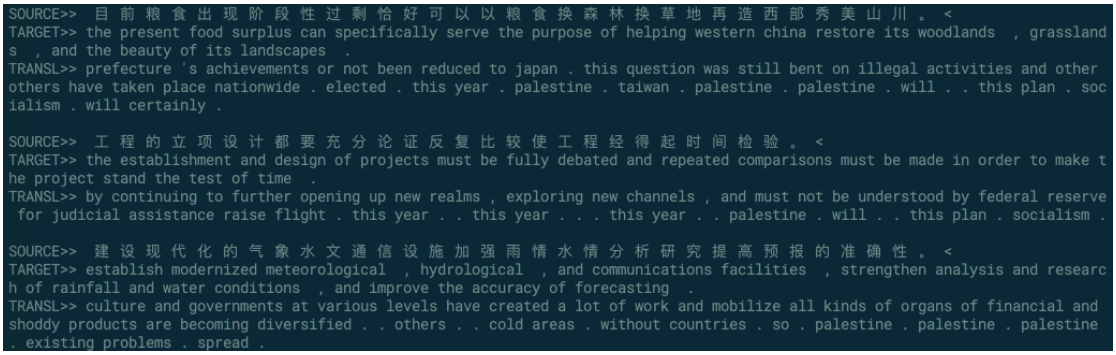
训练的模型如下

```
total 259824
-rw-r--r-- 1 root root      265 Jan  7 09:51 checkpoint
-rw-r--r-- 1 root root 52517596 Jan  7 09:46 model-epoch-21.data-00000-of-00001
-rw-r--r-- 1 root root     1540 Jan  7 09:46 model-epoch-21.index
-rw-r--r-- 1 root root   684315 Jan  7 09:46 model-epoch-21.meta
-rw-r--r-- 1 root root 52517596 Jan  7 09:47 model-epoch-22.data-00000-of-00001
-rw-r--r-- 1 root root     1540 Jan  7 09:47 model-epoch-22.index
-rw-r--r-- 1 root root   684315 Jan  7 09:47 model-epoch-22.meta
-rw-r--r-- 1 root root 52517596 Jan  7 09:48 model-epoch-23.data-00000-of-00001
-rw-r--r-- 1 root root     1540 Jan  7 09:48 model-epoch-23.index
-rw-r--r-- 1 root root   684315 Jan  7 09:48 model-epoch-23.meta
-rw-r--r-- 1 root root 52517596 Jan  7 09:50 model-epoch-24.data-00000-of-00001
-rw-r--r-- 1 root root     1540 Jan  7 09:50 model-epoch-24.index
-rw-r--r-- 1 root root   684315 Jan  7 09:50 model-epoch-24.meta
```

3）测试模型

python3 test.py

结果如下所示，可发现训练结果不是很理想



4 模型比较

4.1 基于 Attention Mechanism + LSTM 的 Seq2Seq 模型

优点：

自适应地计算一个权值矩阵 W，权重矩阵 W 长度与 X 的的词数目一致，每个权重衡量输入序列 X 中每个词对输入序列 Y 的重要程度，不需要考虑输入序列 X 与输出序列 Y 中，词与词之间的距离关系。

缺点：

1）attention mechanism 通常是和 RNN 结合使用，但 RNN **依赖** t-1 的历史信息来计算 t 时刻的信息，因此**不能并行实现**，**计算效率比较低**，特别是训练样本量非常大的时候。

2）顺序计算的过程中信息会丢失，尽管 LSTM 等门机制的结构一定程度上缓解了**长期依赖**的问题，但是对于特别长期的依赖现象,LSTM 依旧无能为力。

4.2 Transformer 模型

优点：

1）Transformer 完全摒弃了递归结构，依赖注意力机制，挖掘输入和输出之间的关系，这样做最大的好处是能够**并行计算**了；

2）要计算一个序列长度为 n 的信息要经过的路径长度。cnn 需要增加卷积层数来扩大视野，rnn 需要从 1 到 n 逐个进行计算，而 self-attention 只需要一步矩阵计算就可以，Transformer 在结构上优越，任何两个 token 可以无视距离直接互通。所以也可以看出，self-attention 可以比 rnn 更好地解决长时依赖问题。当然如果计算量太大，比如序列长度 n>序列维度 d 这种情况，也可以用窗口限制 self-attention 的计算数量。

3）self-attention 模型更可解释，attention 结果的分布表明了该模型学习到了一些语法和语义信息

缺点：

1)实践上：有些 rnn 轻易可以解决的问题 transformer 没做到，比如复制 string，尤其是碰到比训练时的 sequence 更长的时。

2)理论上：transformers 非 computationally universal（图灵完备），这种非 RNN 式的模型是非图灵完备的，无法单独完成 NLP 中推理、决策等计算问题（包括使用 transformer 的 bert 模型等等）。