机器翻译实践

玖强

Outline

□ NMT最新进展

□ NMT工程实践-OpenNMT

□ NMT扩展—股票预测

更多的改进。。。

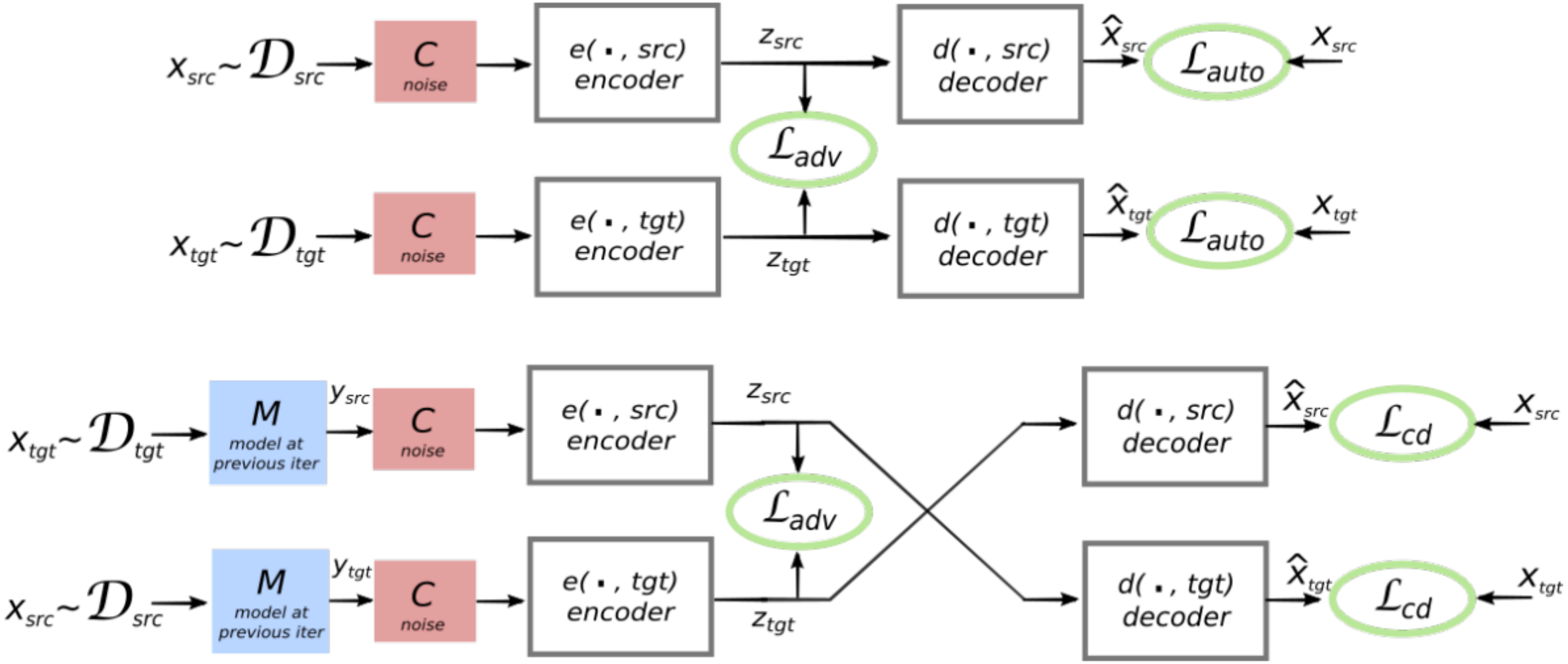
问题和改进方法

- 没数据怎么办？
  - 无监督，unpaired学习
  - Pivot learning
- 训练和测试不匹配怎么办？
  - 训练时我们用Schedule sampling, 测试的时候我们用beam search
  - 训练的时候我们用cross-entropy, 测试的时候我们用BLEU评价

没数据怎么办

Unsupervised Machine Translation Using Monolingual Corpora Only

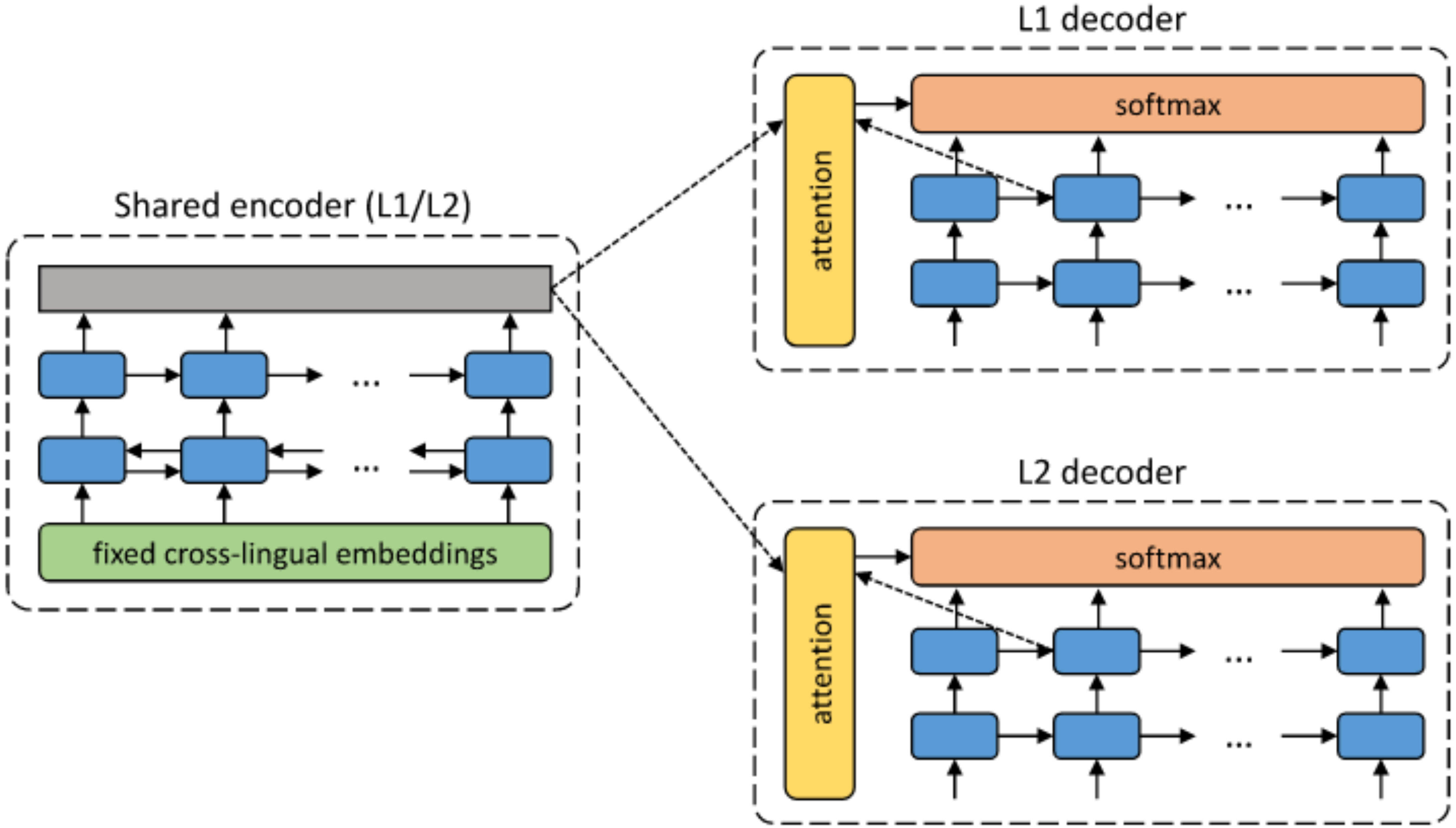
Lample, Guillaume, Ludovic Denoyer, and Marc'Aurelio Ranzato. "Unsupervised Machine Translation Using Monolingual Corpora Only." arXiv preprint arXiv:1711.00043 (2017).



https://rawgit.com/gujiuxiang/PaperNotes/master/post/Recurrent_Neural_Network/20171031_Unsupervised_Machine_Translation_Using_Monolingual_Corpora_Only.md.html

Unsupervised Neural Machine Translation

Artetxe, Mikel, et al. "Unsupervised neural machine translation." arXiv preprint arXiv:1710.11041 (2017).

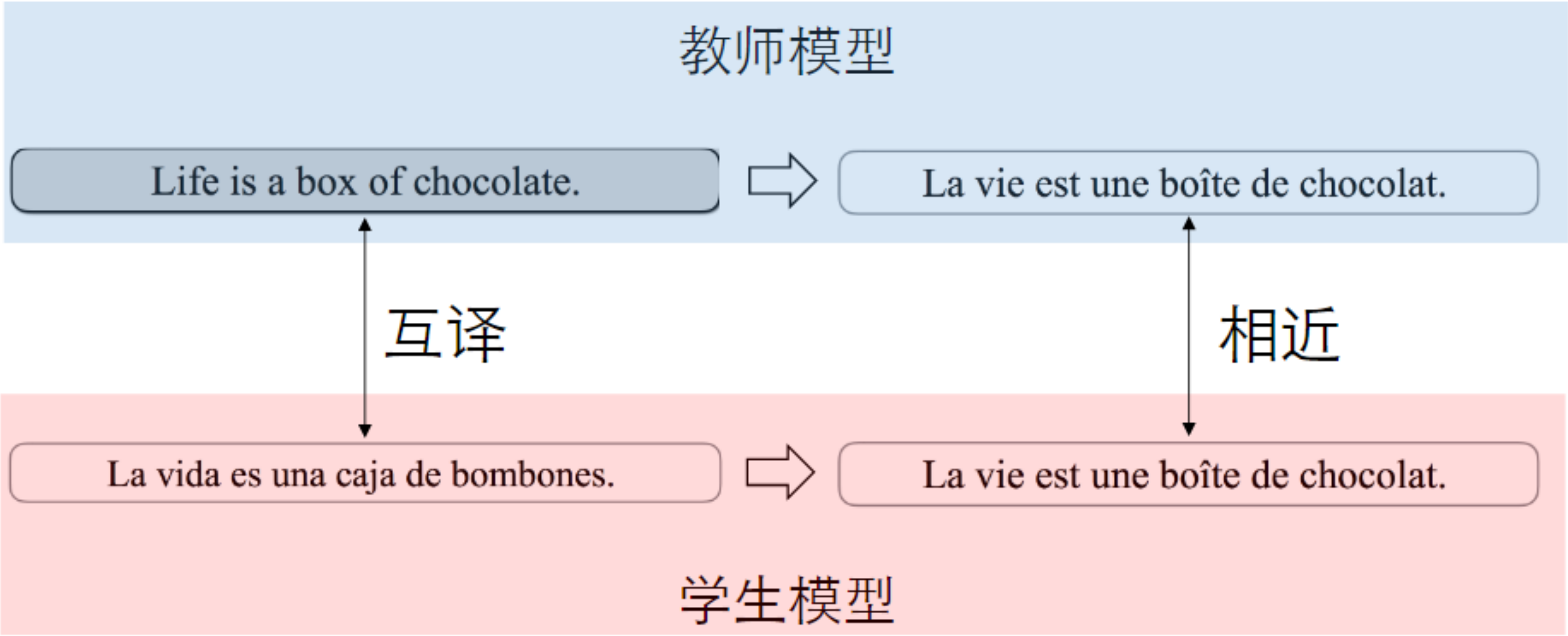


https://rawgit.com/gujiuxiang/PaperNotes/master/post/Recurrent_Neural_Network/20171030_Unsupervised_Neural_Machine_Translation.md.html

零资源语言翻译

❑ 面面向零资源翻译的“教师-学生”框架

Chen, Yun, et al. "A Teacher-Student Framework for Zero-Resource Neural Machine Translation." arXiv preprint arXiv:1705.00753 (2017).


教师模型
Life is a box of chocolate.
La vie est une boîte de chocolat.
互译
相近
La vida es una caja de bombones.
La vie est une boîte de chocolat.
学生模型

$$\mathcal{J}_{\text{SENT}}(\boldsymbol{\theta}_{x\to y}) = \sum_{\langle \mathbf{x},\mathbf{z}\rangle \in D_{x,z}} \text{KL}\Big(P(\mathbf{y}|\mathbf{z};\hat{\boldsymbol{\theta}}_{z\to y}) \Big|\Big| P(\mathbf{y}|\mathbf{x};\boldsymbol{\theta}_{x\to y})\Big)$$

https://rawgit.com/gujiuxiang/PaperNotes/master/post/Recurrent_Neural_Network/20170502_A_Teacher-Student_Framework_for_Zero-Resource_Neural_Machine_Translation.md.html

Interactive Attention for Neural Machine Translation

Meng, Fandong, et al. "Interactive attention for neural machine translation." arXiv preprint arXiv:1610.05011 (2016).

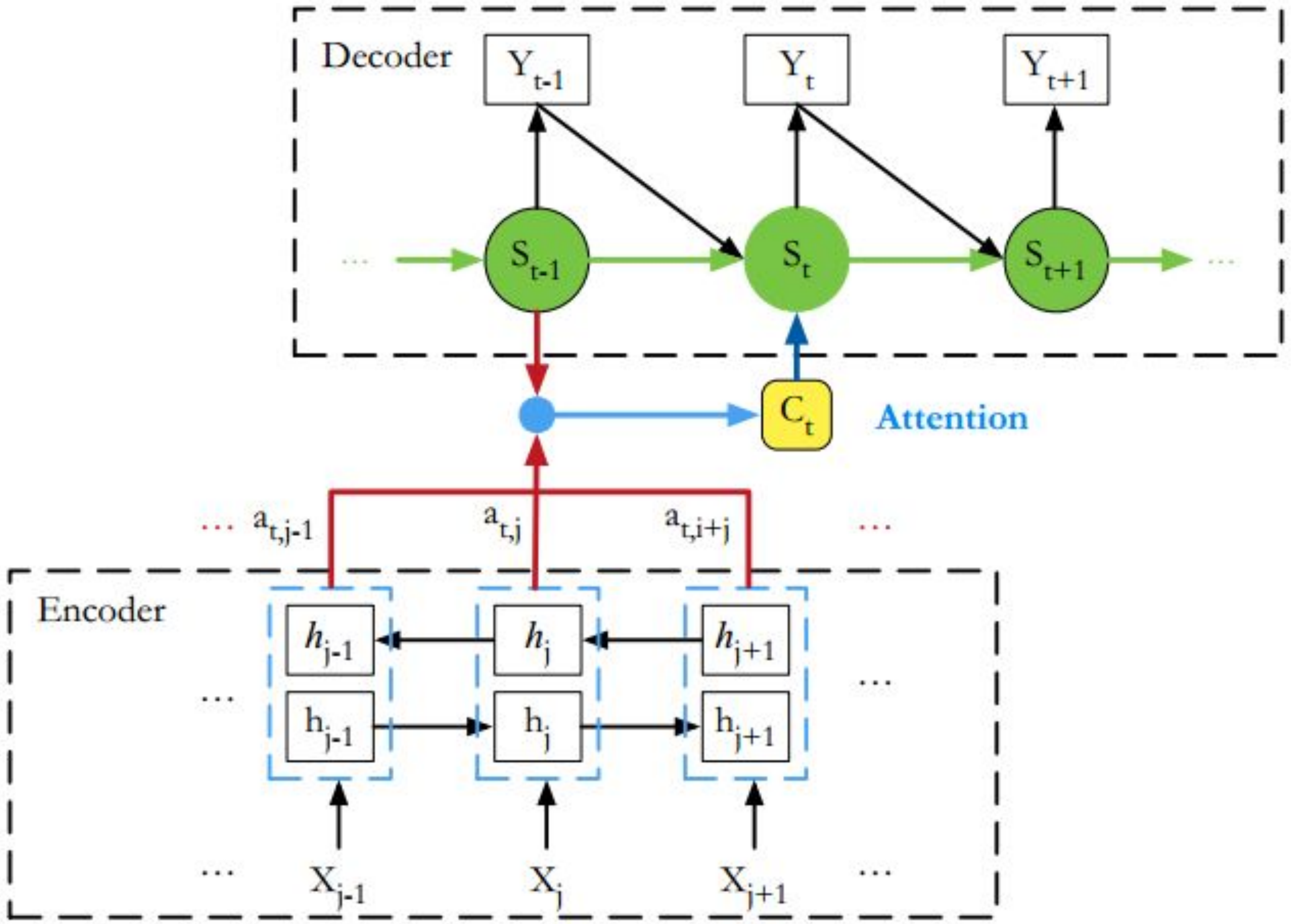


Figure 1: Illustration for attention-based NMT.

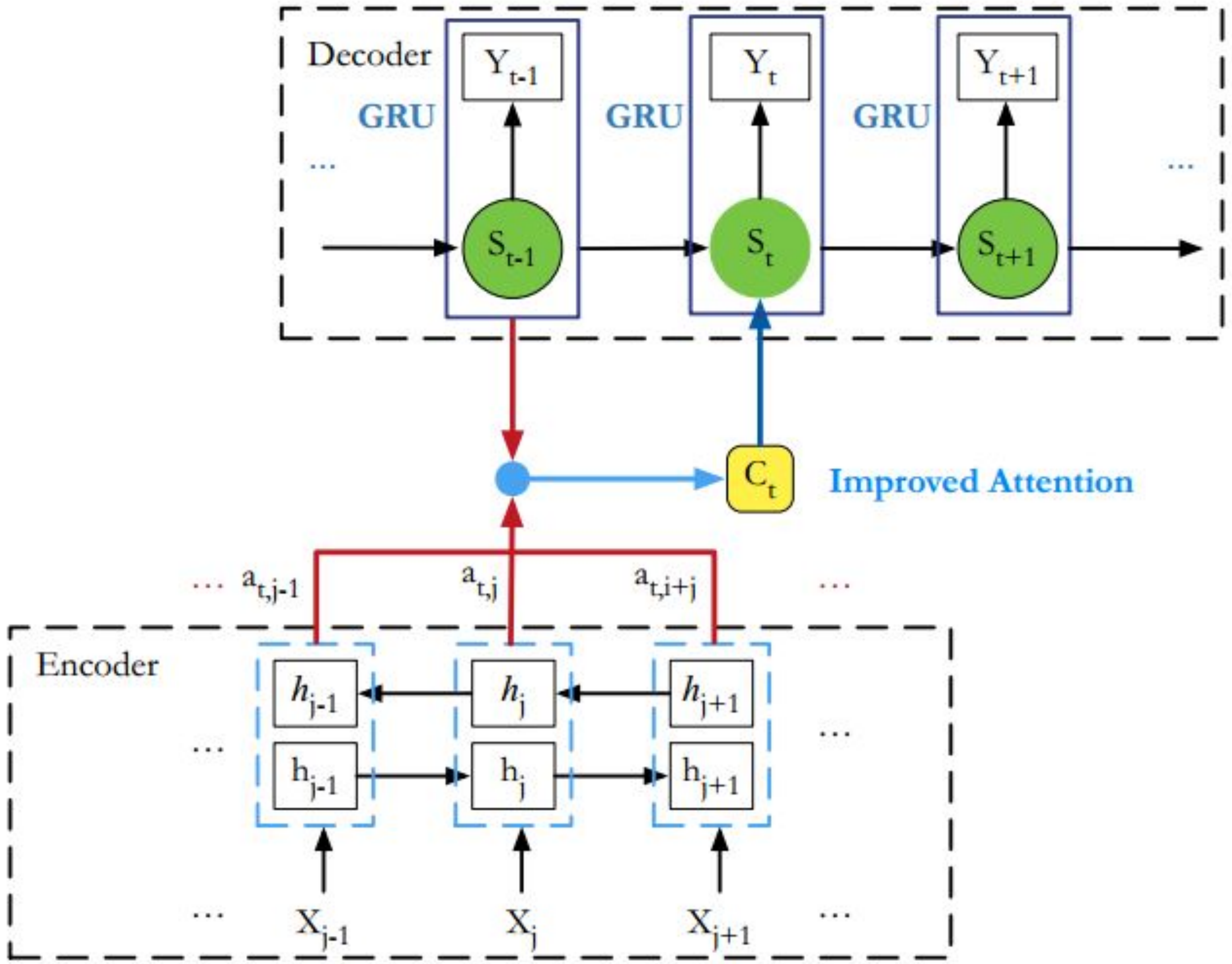
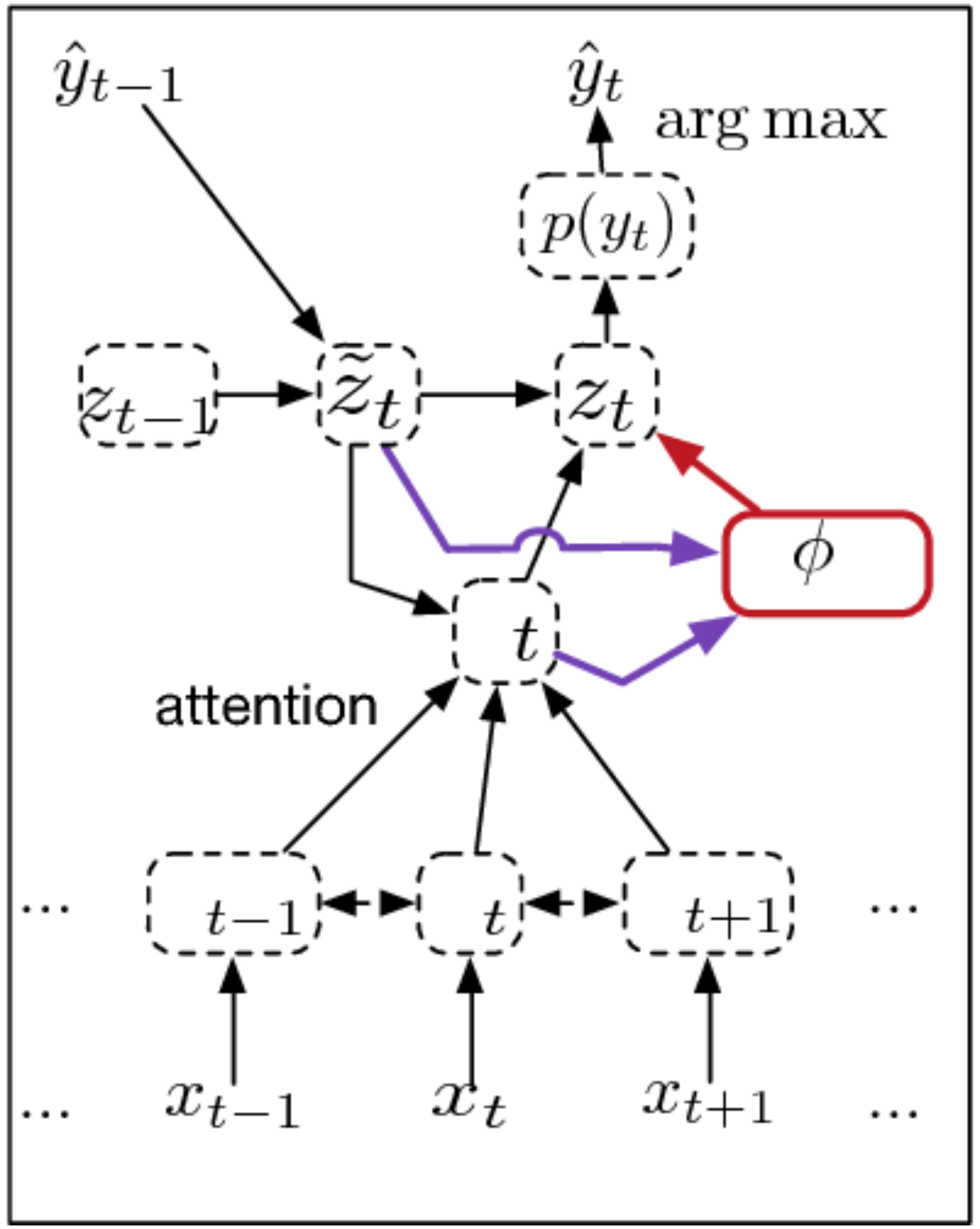


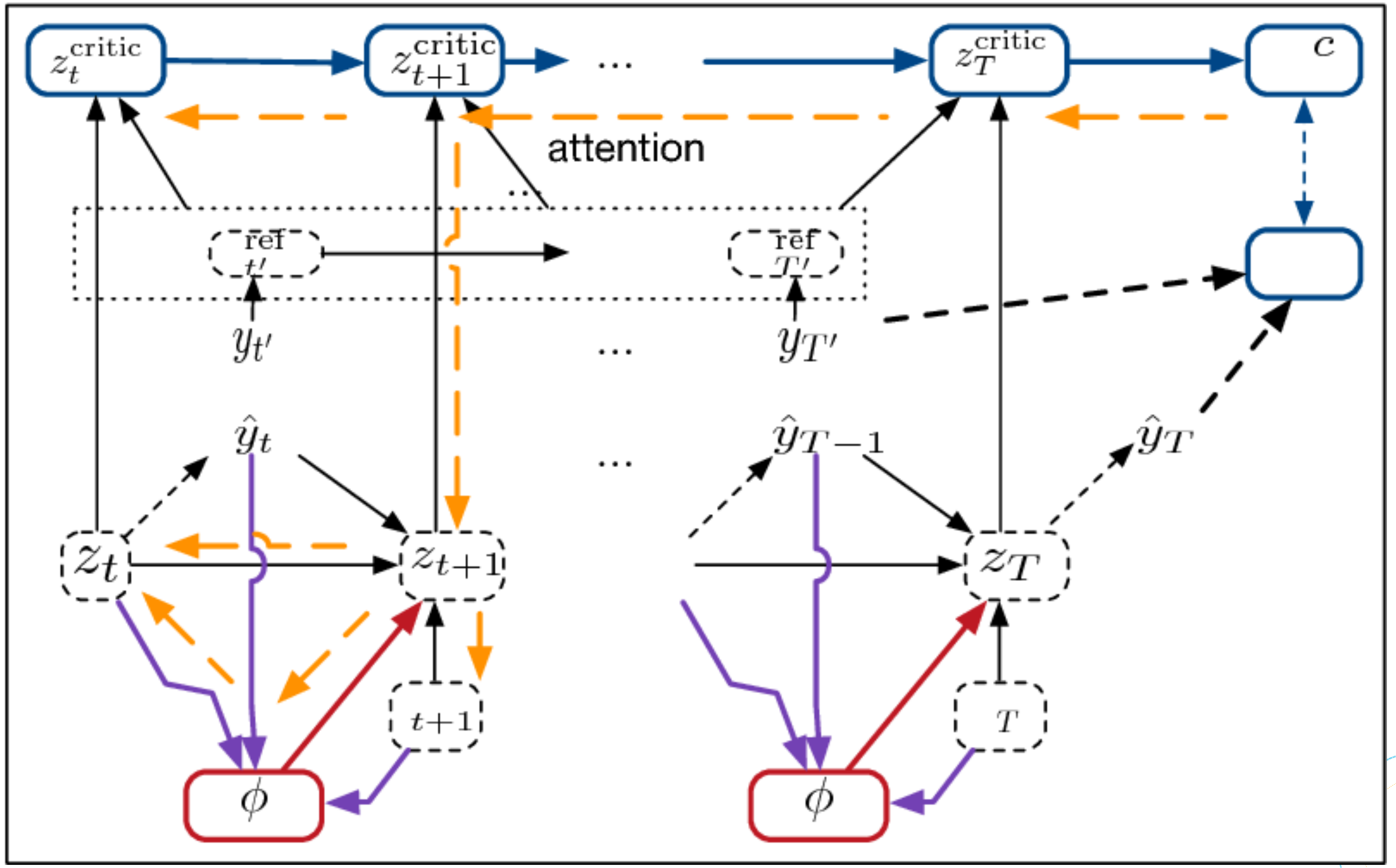
Figure 2: Illustration for improved attention model of NMT.

训练测试不匹配怎么办

Trainable Greedy Decoding for Neural Machine Translation

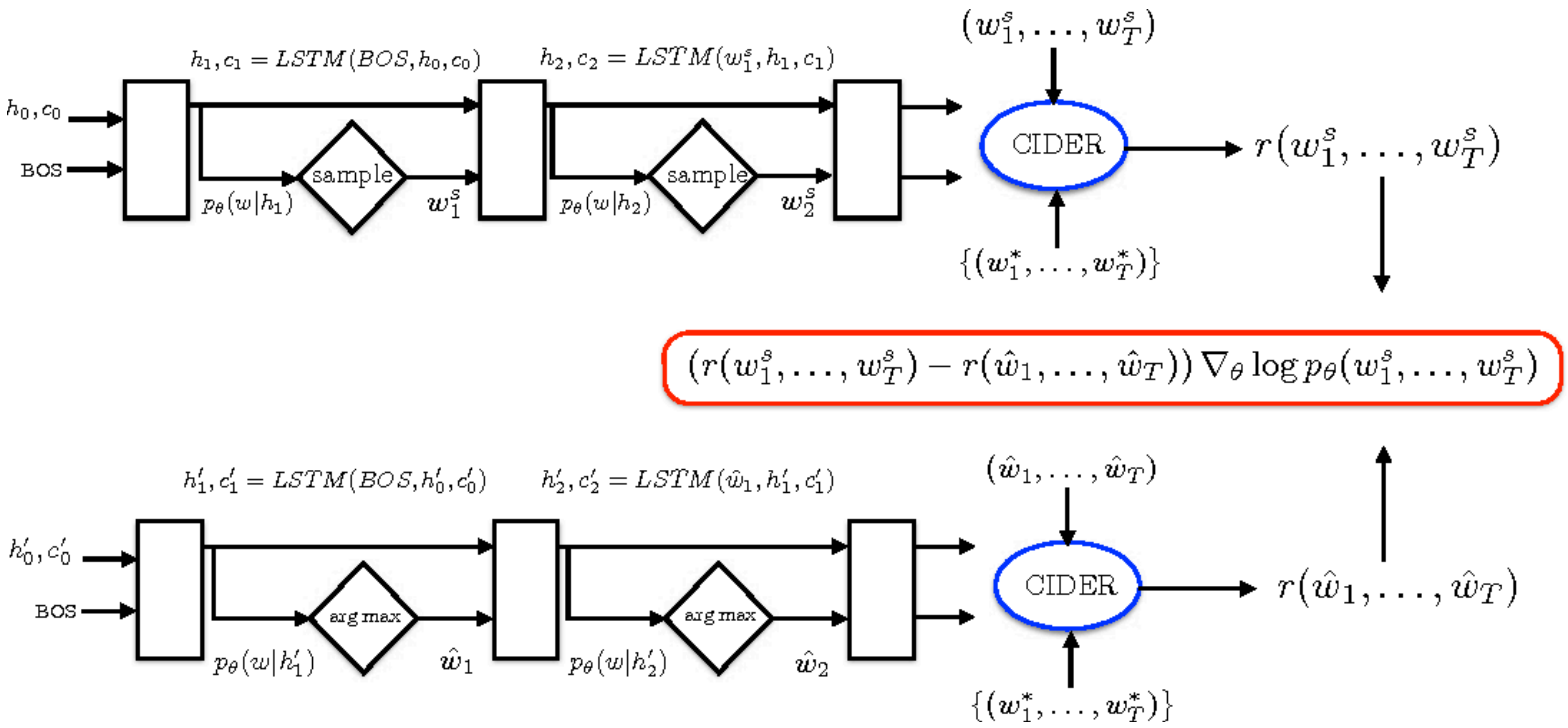
Gu, Jiatao, Kyunghyun Cho, and Victor OK Li. "Trainable greedy decoding for neural machine translation." arXiv preprint arXiv:1702.02429 (2017).





SELF-CRITICAL LEARNING IMAGE CAPTIONING

Rennie, Steven J., et al. "Self-critical sequence training for image captioning." *arXiv preprint arXiv:1612.00563* (2016).

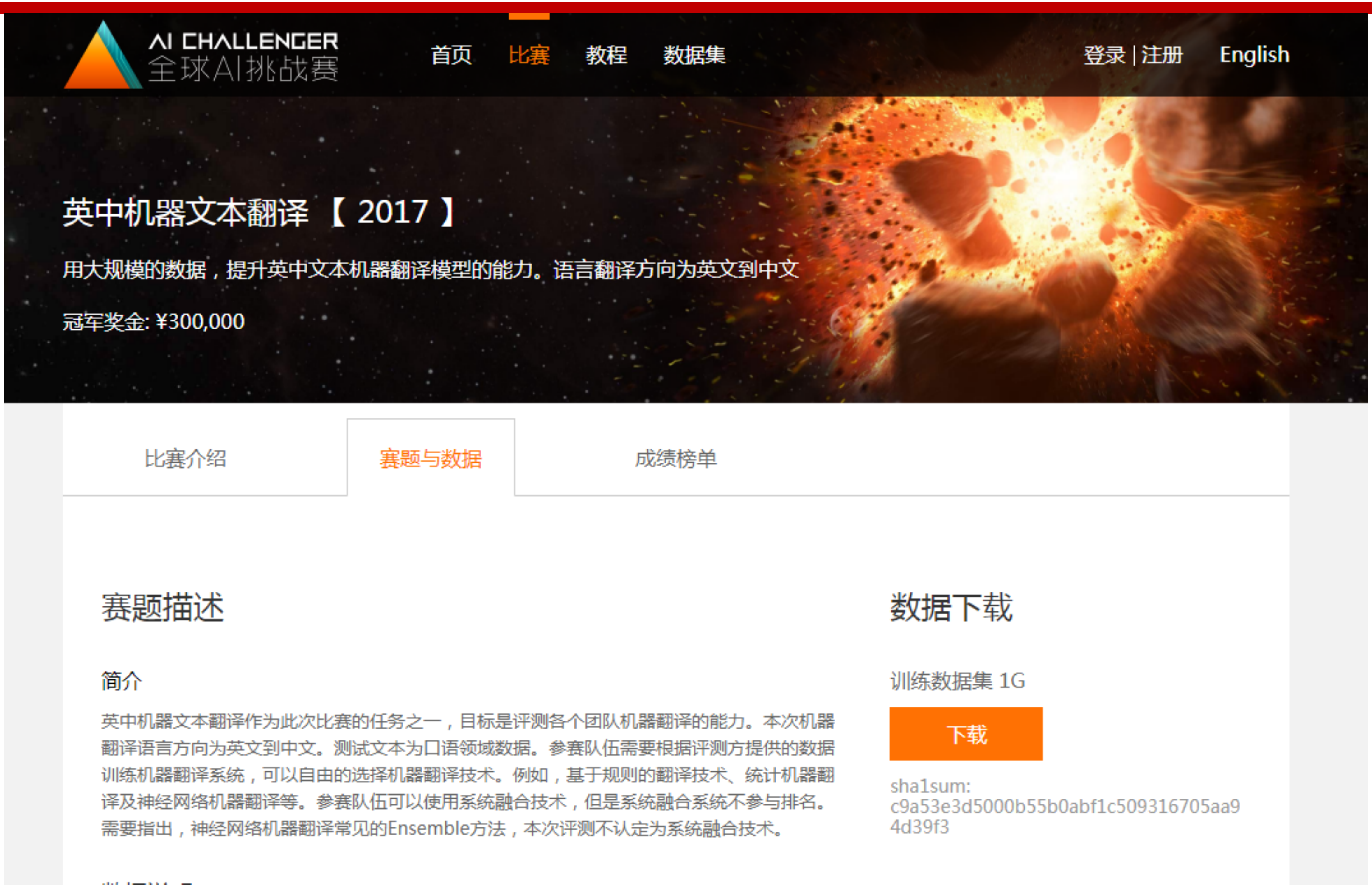


$h_1, c_1 = LSTM(BOS, h_0, c_0)$
$h_2, c_2 = LSTM(w_1^s, h_1, c_1)$
$(w_1^s, \ldots, w_T^s)$
h_0, c_0
BOS
sample
$p_\theta(w|h_1)$
w_1^s
sample
$p_\theta(w|h_2)$
w_2^s
CIDER
$r(w_1^s, \ldots, w_T^s)$
$\{(w_1^*, \ldots, w_T^*)\}$
$(r(w_1^s, \ldots, w_T^s) - r(\hat{w}_1, \ldots, \hat{w}_T)) \nabla_\theta \log p_\theta(w_1^s, \ldots, w_T^s)$
$h_1', c_1' = LSTM(BOS, h_0', c_0')$
$h_2', c_2' = LSTM(\hat{w}_1, h_1', c_1')$
$(\hat{w}_1, \ldots, \hat{w}_T)$
h_0', c_0'
BOS
arg max
$p_\theta(w|h_1')$
$\hat{w}_1$
arg max
$p_\theta(w|h_2')$
$\hat{w}_2$
CIDER
$r(\hat{w}_1, \ldots, \hat{w}_T)$
$\{(w_1^*, \ldots, w_T^*)\}$

Seq2seq实践—中英翻译

OpenNMT: Open-Source Toolkit for Neural Machine Translation

数据库



英中机器文本翻译 【 2017 】

用大规模的数据，提升英中文本机器翻译模型的能力。语言翻译方向为英文到中文

冠军奖金: ¥300,000

赛题描述

简介

英中机器文本翻译作为此次比赛的任务之一，目标是评测各个团队机器翻译的能力。本次机器翻译语言方向为英文到中文。测试文本为口语领域数据。参赛队伍需要根据评测方提供的数据训练机器翻译系统，可以自由的选择机器翻译技术。例如，基于规则的翻译技术、统计机器翻译及神经网络机器翻译等。参赛队伍可以使用系统融合技术，但是系统融合系统不参与排名。需要指出，神经网络机器翻译常见的Ensemble方法，本次评测不认定为系统融合技术。

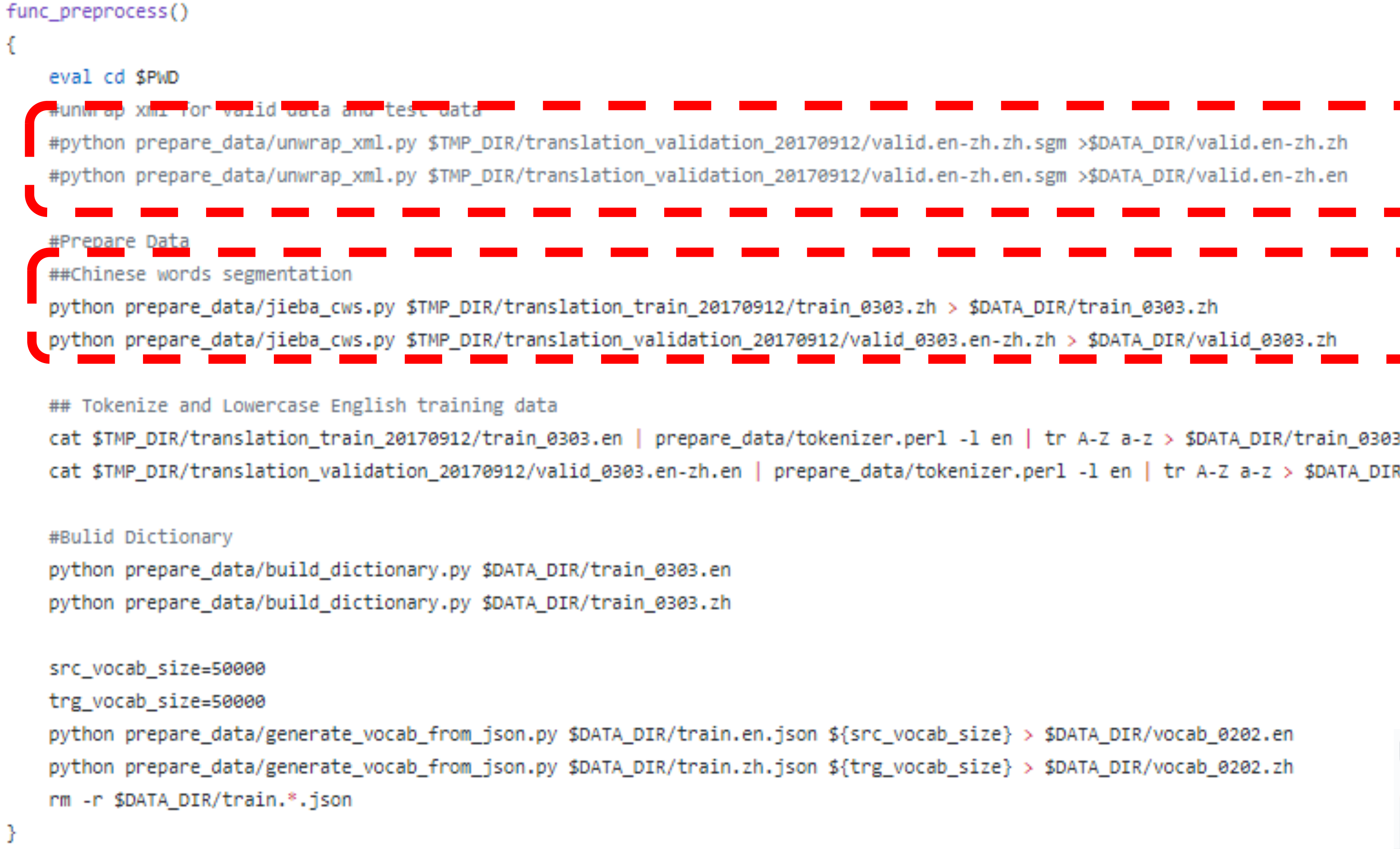
数据下载

训练数据集 1G

sha1sum:
c9a53e3d5000b55b0abf1c509316705aa9
4d39f3

https://challenger.ai/competition/translation/subject

数据预处理

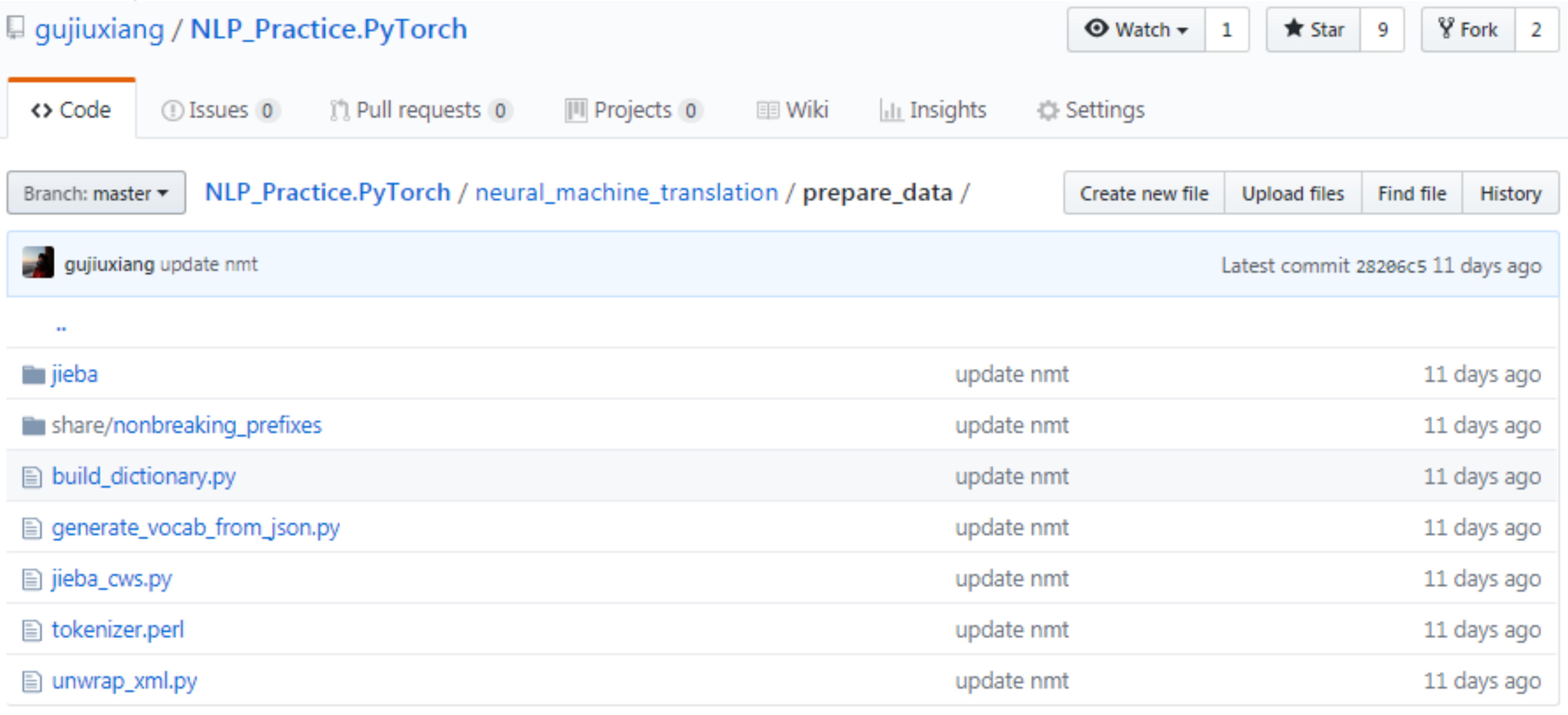


```
func_preprocess()
{
    eval cd $PWD
    #unwrap xml for valid data and test data
    #python prepare_data/unwrap_xml.py $TMP_DIR/translation_validation_20170912/valid.en-zh.zh.sgm >$DATA_DIR/valid.en-zh.zh
    #python prepare_data/unwrap_xml.py $TMP_DIR/translation_validation_20170912/valid.en-zh.en.sgm >$DATA_DIR/valid.en-zh.en

    #Prepare Data
    ##Chinese words segmentation
    python prepare_data/jieba_cws.py $TMP_DIR/translation_train_20170912/train_0303.zh > $DATA_DIR/train_0303.zh
    python prepare_data/jieba_cws.py $TMP_DIR/translation_validation_20170912/valid_0303.en-zh.zh > $DATA_DIR/valid_0303.zh

    ## Tokenize and Lowercase English training data
    cat $TMP_DIR/translation_train_20170912/train_0303.en | prepare_data/tokenizer.perl -l en | tr A-Z a-z > $DATA_DIR/train_0303.en
    cat $TMP_DIR/translation_validation_20170912/valid_0303.en-zh.en | prepare_data/tokenizer.perl -l en | tr A-Z a-z > $DATA_DIR/valid_030

    #Bulid Dictionary
    python prepare_data/build_dictionary.py $DATA_DIR/train_0303.en
    python prepare_data/build_dictionary.py $DATA_DIR/train_0303.zh

    src_vocab_size=50000
    trg_vocab_size=50000
    python prepare_data/generate_vocab_from_json.py $DATA_DIR/train.en.json ${src_vocab_size} > $DATA_DIR/vocab_0202.en
    python prepare_data/generate_vocab_from_json.py $DATA_DIR/train.zh.json ${trg_vocab_size} > $DATA_DIR/vocab_0202.zh
    rm -r $DATA_DIR/train.*.json
}
```

数据格式转换

中文分词

英文处理



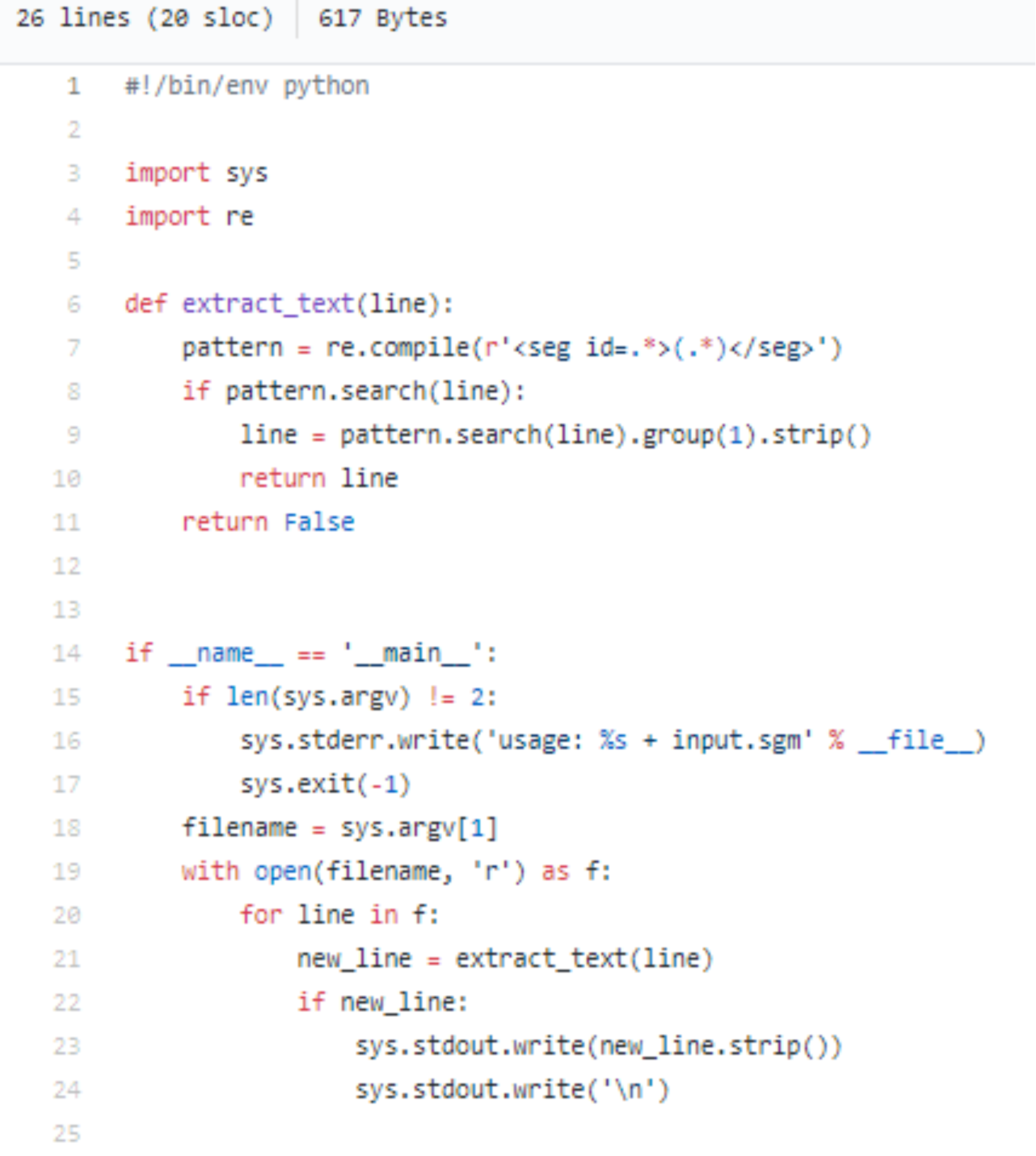
gujiuxiang / NLP_Practice.PyTorch

NLP_Practice.PyTorch / neural_machine_translation / prepare_data /

gujiuxiang update nmt — Latest commit 28206c5 11 days ago

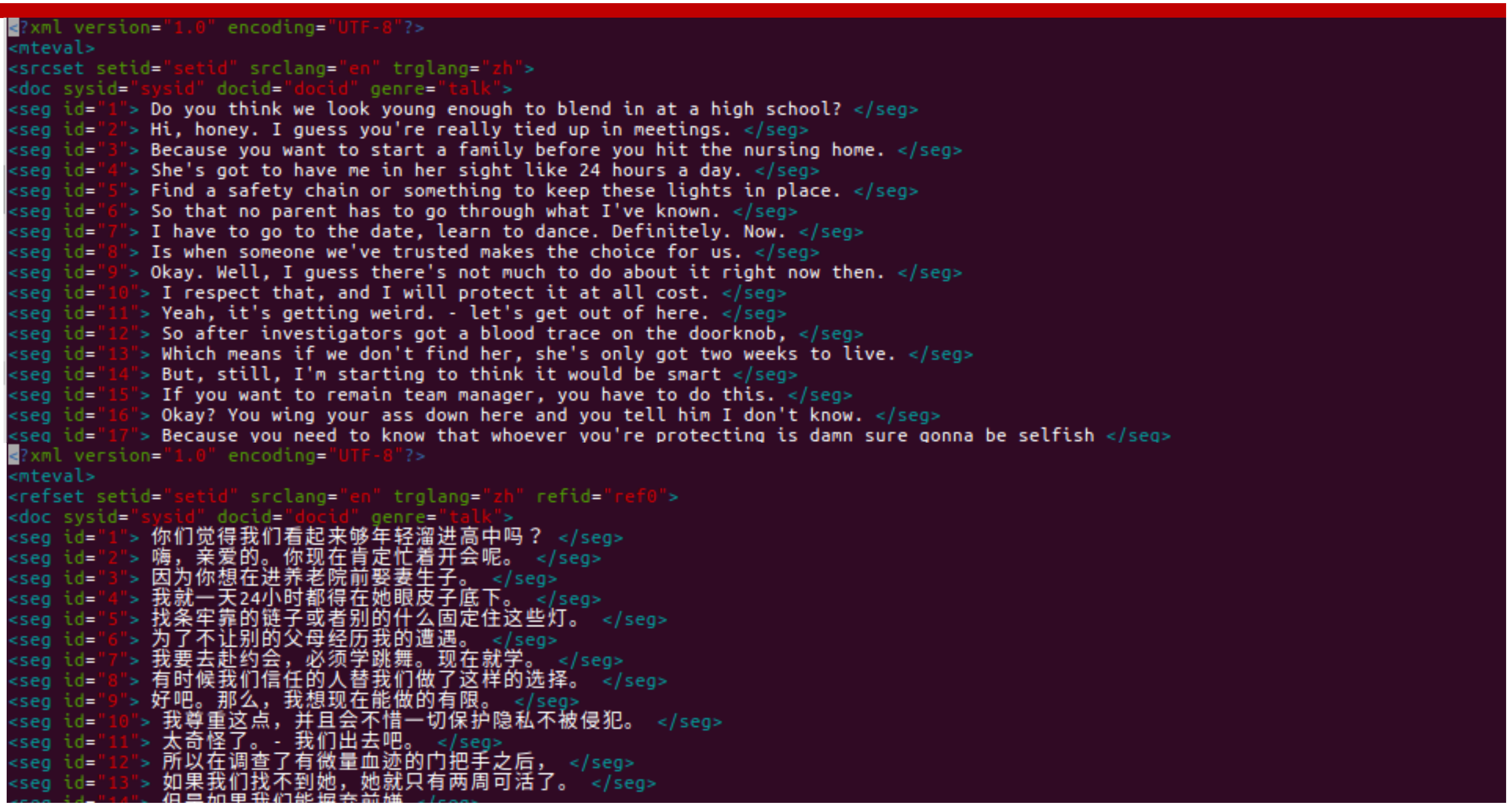
..		
jieba	update nmt	11 days ago
share/nonbreaking_prefixes	update nmt	11 days ago
build_dictionary.py	update nmt	11 days ago
generate_vocab_from_json.py	update nmt	11 days ago
jieba_cws.py	update nmt	11 days ago
tokenizer.perl	update nmt	11 days ago
unwrap_xml.py	update nmt	11 days ago

数据预处理—格式转换

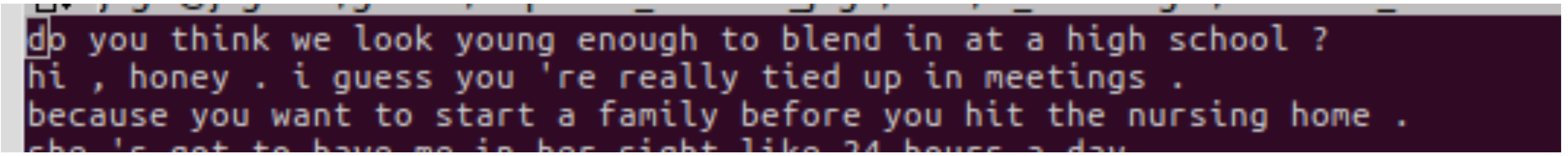


26 lines (20 sloc)   617 Bytes

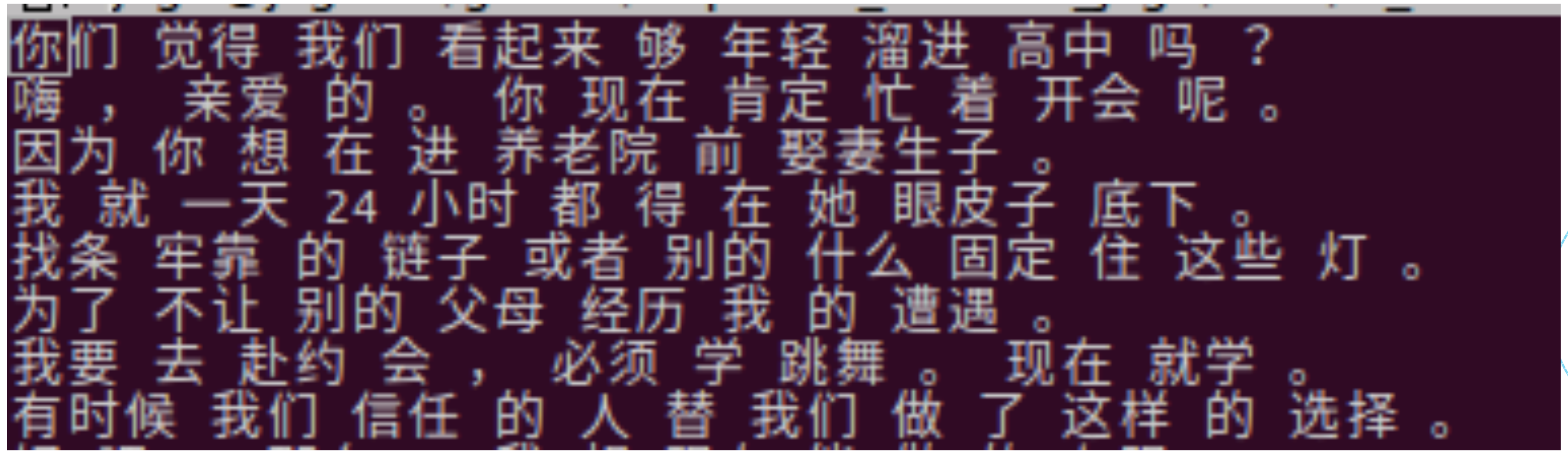
```python
#!/bin/env python

import sys
import re

def extract_text(line):
    pattern = re.compile(r'<seg id=.*>(.*)</seg>')
    if pattern.search(line):
        line = pattern.search(line).group(1).strip()
        return line
    return False


if __name__ == '__main__':
    if len(sys.argv) != 2:
        sys.stderr.write('usage: %s + input.sgm' % __file__)
        sys.exit(-1)
    filename = sys.argv[1]
    with open(filename, 'r') as f:
        for line in f:
            new_line = extract_text(line)
            if new_line:
                sys.stdout.write(new_line.strip())
                sys.stdout.write('\n')

```



```
<?xml version="1.0" encoding="UTF-8"?>
<mteval>
<srcset setid="setid" srclang="en" trglang="zh">
<doc sysid="sysid" docid="docid" genre="talk">
<seg id="1"> Do you think we look young enough to blend in at a high school? </seg>
<seg id="2"> Hi, honey. I guess you're really tied up in meetings. </seg>
<seg id="3"> Because you want to start a family before you hit the nursing home. </seg>
<seg id="4"> She's got to have me in her sight like 24 hours a day. </seg>
<seg id="5"> Find a safety chain or something to keep these lights in place. </seg>
<seg id="6"> So that no parent has to go through what I've known. </seg>
<seg id="7"> I have to go to the date, learn to dance. Definitely. Now. </seg>
<seg id="8"> Is when someone we've trusted makes the choice for us. </seg>
<seg id="9"> Okay. Well, I guess there's not much to do about it right now then. </seg>
<seg id="10"> I respect that, and I will protect it at all cost. </seg>
<seg id="11"> Yeah, it's getting weird. - let's get out of here. </seg>
<seg id="12"> So after investigators got a blood trace on the doorknob, </seg>
<seg id="13"> Which means if we don't find her, she's only got two weeks to live. </seg>
<seg id="14"> But, still, I'm starting to think it would be smart </seg>
<seg id="15"> If you want to remain team manager, you have to do this. </seg>
<seg id="16"> Okay? You wing your ass down here and you tell him I don't know. </seg>
<seg id="17"> Because you need to know that whoever you're protecting is damn sure gonna be selfish </seg>
<?xml version="1.0" encoding="UTF-8"?>
<mteval>
<refset setid="setid" srclang="en" trglang="zh" refid="ref0">
<doc sysid="sysid" docid="docid" genre="talk">
<seg id="1"> 你们觉得我们看起来够年轻溜进高中吗？ </seg>
<seg id="2"> 嗨，亲爱的。你现在肯定忙着开会呢。 </seg>
<seg id="3"> 因为你想在进养老院前娶妻生子。 </seg>
<seg id="4"> 我就一天24小时都得在她眼皮子底下。 </seg>
<seg id="5"> 找条牢靠的链子或者别的什么固定住这些灯。 </seg>
<seg id="6"> 为了不让别的父母经历我的遭遇。 </seg>
<seg id="7"> 我要去赴约会，必须学跳舞。现在就学。 </seg>
<seg id="8"> 有时候我们信任的人替我们做了这样的选择。 </seg>
<seg id="9"> 好吧。那么，我想现在能做的有限。 </seg>
<seg id="10"> 我尊重这点，并且会不惜一切保护隐私不被侵犯。 </seg>
<seg id="11"> 太奇怪了。- 我们出去吧。 </seg>
<seg id="12"> 所以在调查了有微量血迹的门把手之后， </seg>
<seg id="13"> 如果我们找不到她，她就只有两周可活了。 </seg>
<seg id="14"> 但是如果我们能摒弃前嫌 </seg>
```



```
do you think we look young enough to blend in at a high school ?
hi , honey . i guess you 're really tied up in meetings .
because you want to start a family before you hit the nursing home .
she 's got to have me in her sight like 24 hours a day
```



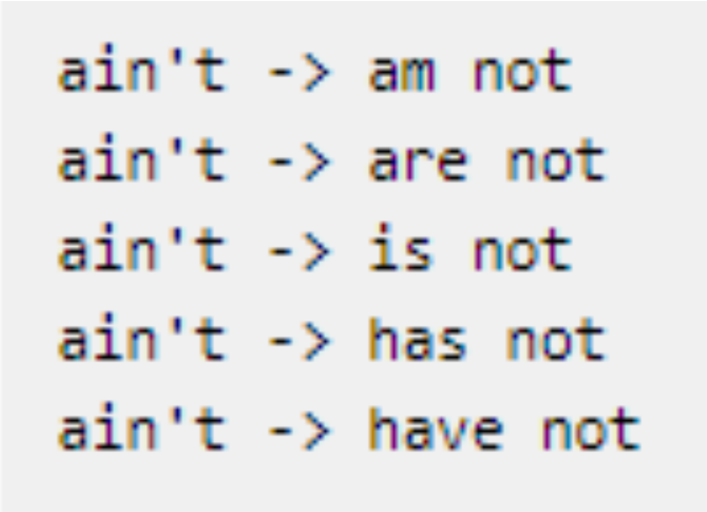
```
你们 觉得 我们 看起来 够 年轻 溜进 高中 吗 ？
嗨 ， 亲爱 的 。 你 现在 肯定 忙 着 开会 呢 。
因为 你 想 在 进 养老院 前 娶妻生子 。
我 就 一天 24 小时 都 得 在 她 眼皮子 底下 。
找条 牢靠 的 链子 或者 别的 什么 固定 住 这些 灯 。
为了 不让 别的 父母 经历 我 的 遭遇 。
我要 去 赴约 会 ， 必须 学 跳舞 。 现在 就学 。
有时候 我们 信任 的 人 替 我们 做 了 这样 的 选择 。
```

数据预处理—格式转换

```
import contractions
…
 _sline = re.sub(ur"[%s]+" %punctuation, "", _sline.decode("utf-8"))
srcF_filter.append(_sline)
tmp_tline = contractions.fix(_tline)
tmp_tline = clean_en_punctuations(clean_en_string(tmp_tline))
tgtF_filter.append(tmp_tline.lower())
```

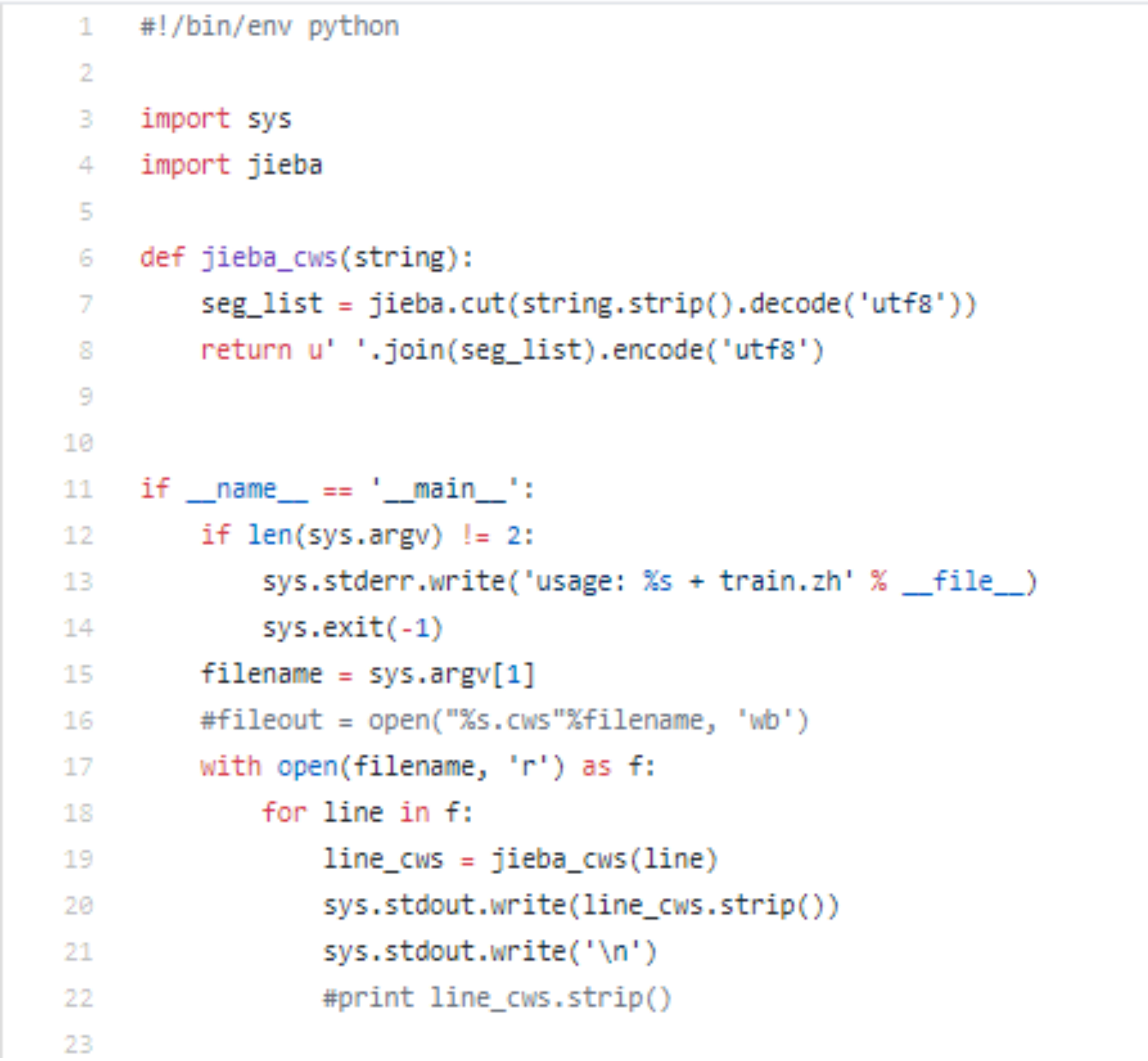
https://pypi.python.org/pypi/pycontractions/1.0.1

Expanding English language contractions in Python



```
ain't -> am not
ain't -> are not
ain't -> is not
ain't -> has not
ain't -> have not
```

数据预处理--中文分词



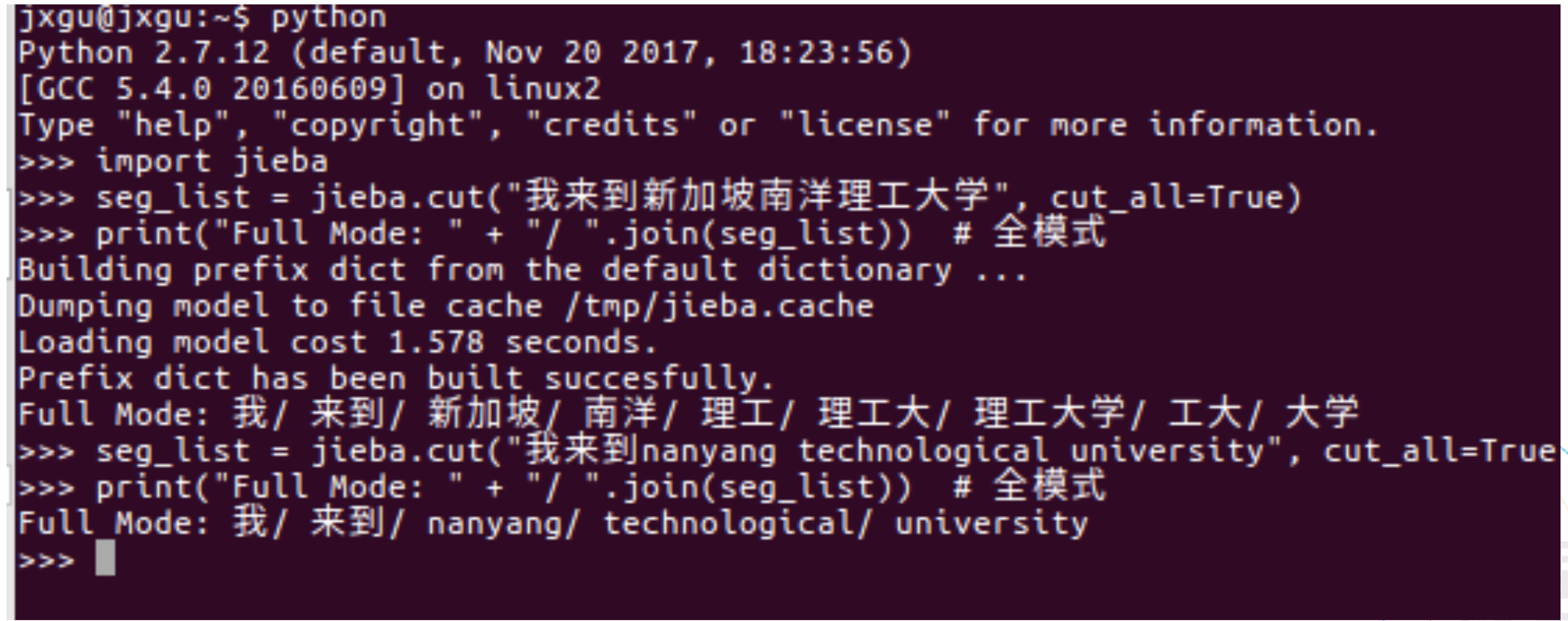
```python
#!/bin/env python

import sys
import jieba

def jieba_cws(string):
    seg_list = jieba.cut(string.strip().decode('utf8'))
    return u' '.join(seg_list).encode('utf8')


if __name__ == '__main__':
    if len(sys.argv) != 2:
        sys.stderr.write('usage: %s + train.zh' % __file__)
        sys.exit(-1)
    filename = sys.argv[1]
    #fileout = open("%s.cws"%filename, 'wb')
    with open(filename, 'r') as f:
        for line in f:
            line_cws = jieba_cws(line)
            sys.stdout.write(line_cws.strip())
            sys.stdout.write('\n')
            #print line_cws.strip()

```

pip install jieba / pip3 install jieba

支持三种分词模式：

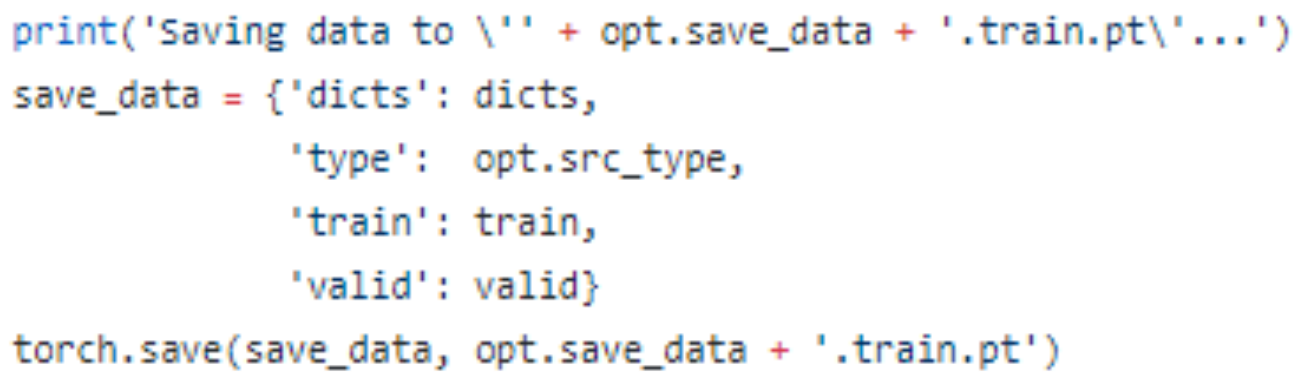
1. 精确模式，试图将句子最精确地切开，适合文本分析；
2. 全模式，把句子中所有的可以成词的词语都扫描出来, 速度非常快，但是不能解决歧义；
3. 搜索引擎模式，在精确模式的基础上，对长词再次切分，提高召回率，适合用于搜索引擎分词。

- jieba.cut(text)
- jieba.cut_for_search(text)



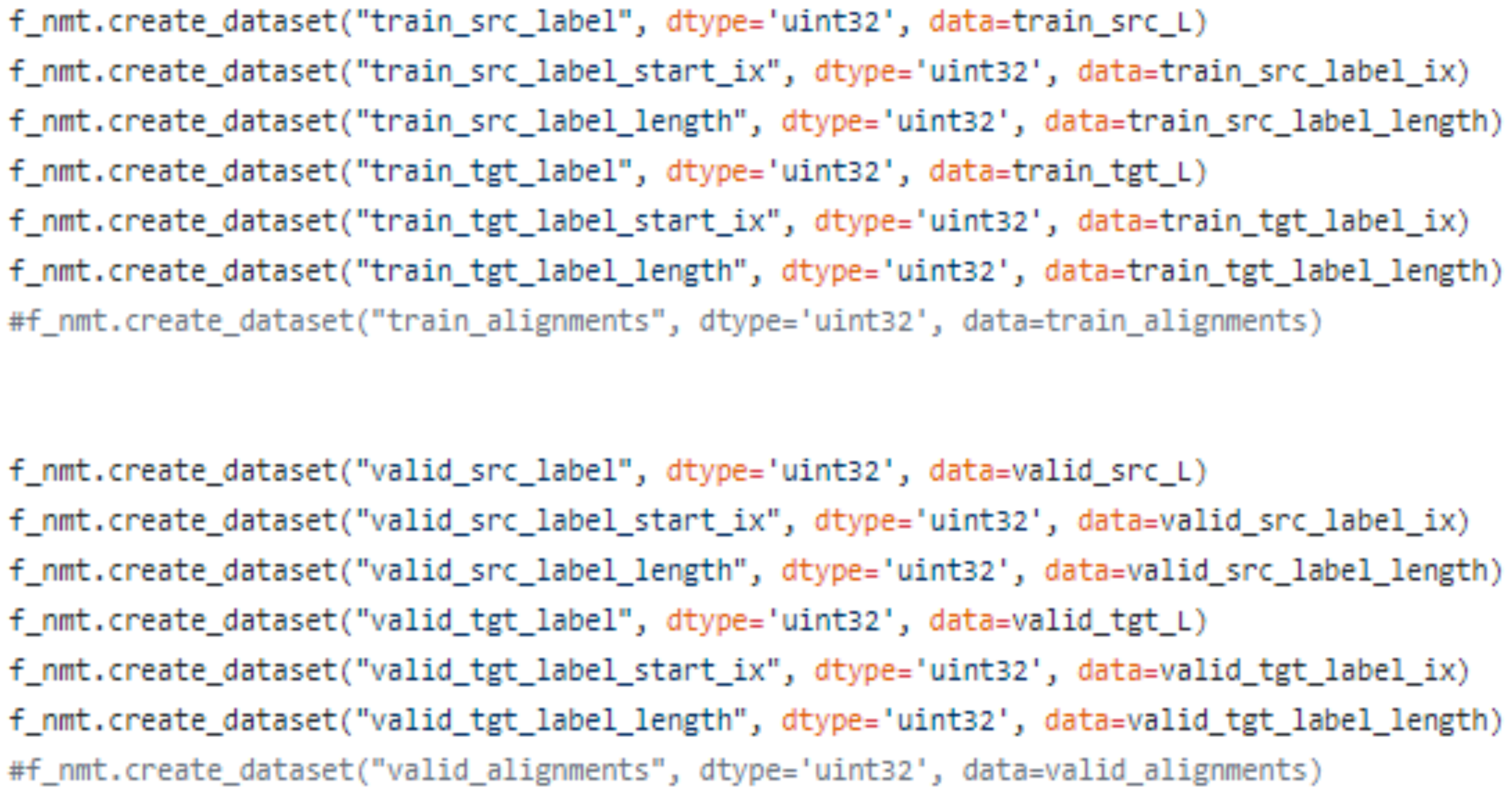
```
jxgu@jxgu:~$ python
Python 2.7.12 (default, Nov 20 2017, 18:23:56)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import jieba
>>> seg_list = jieba.cut("我来到新加坡南洋理工大学", cut_all=True)
>>> print("Full Mode: " + "/ ".join(seg_list))  # 全模式
Building prefix dict from the default dictionary ...
Dumping model to file cache /tmp/jieba.cache
Loading model cost 1.578 seconds.
Prefix dict has been built succesfully.
Full Mode: 我/ 来到/ 新加坡/ 南洋/ 理工/ 理工大/ 理工大学/ 工大/ 大学
>>> seg_list = jieba.cut("我来到nanyang technological university", cut_all=True
>>> print("Full Mode: " + "/ ".join(seg_list))  # 全模式
Full Mode: 我/ 来到/ nanyang/ technological/ university
>>> 
```

数据预处理–创建数据文件



```
print('Saving data to \'' + opt.save_data + '.train.pt\'...')
save_data = {'dicts': dicts,
             'type':  opt.src_type,
             'train': train,
             'valid': valid}
torch.save(save_data, opt.save_data + '.train.pt')
```

Load pt 文件很慢,
需要几十G的内存



```
f_nmt.create_dataset("train_src_label", dtype='uint32', data=train_src_L)
f_nmt.create_dataset("train_src_label_start_ix", dtype='uint32', data=train_src_label_ix)
f_nmt.create_dataset("train_src_label_length", dtype='uint32', data=train_src_label_length)
f_nmt.create_dataset("train_tgt_label", dtype='uint32', data=train_tgt_L)
f_nmt.create_dataset("train_tgt_label_start_ix", dtype='uint32', data=train_tgt_label_ix)
f_nmt.create_dataset("train_tgt_label_length", dtype='uint32', data=train_tgt_label_length)
#f_nmt.create_dataset("train_alignments", dtype='uint32', data=train_alignments)


f_nmt.create_dataset("valid_src_label", dtype='uint32', data=valid_src_L)
f_nmt.create_dataset("valid_src_label_start_ix", dtype='uint32', data=valid_src_label_ix)
f_nmt.create_dataset("valid_src_label_length", dtype='uint32', data=valid_src_label_length)
f_nmt.create_dataset("valid_tgt_label", dtype='uint32', data=valid_tgt_L)
f_nmt.create_dataset("valid_tgt_label_start_ix", dtype='uint32', data=valid_tgt_label_ix)
f_nmt.create_dataset("valid_tgt_label_length", dtype='uint32', data=valid_tgt_label_length)
#f_nmt.create_dataset("valid_alignments", dtype='uint32', data=valid_alignments)
```

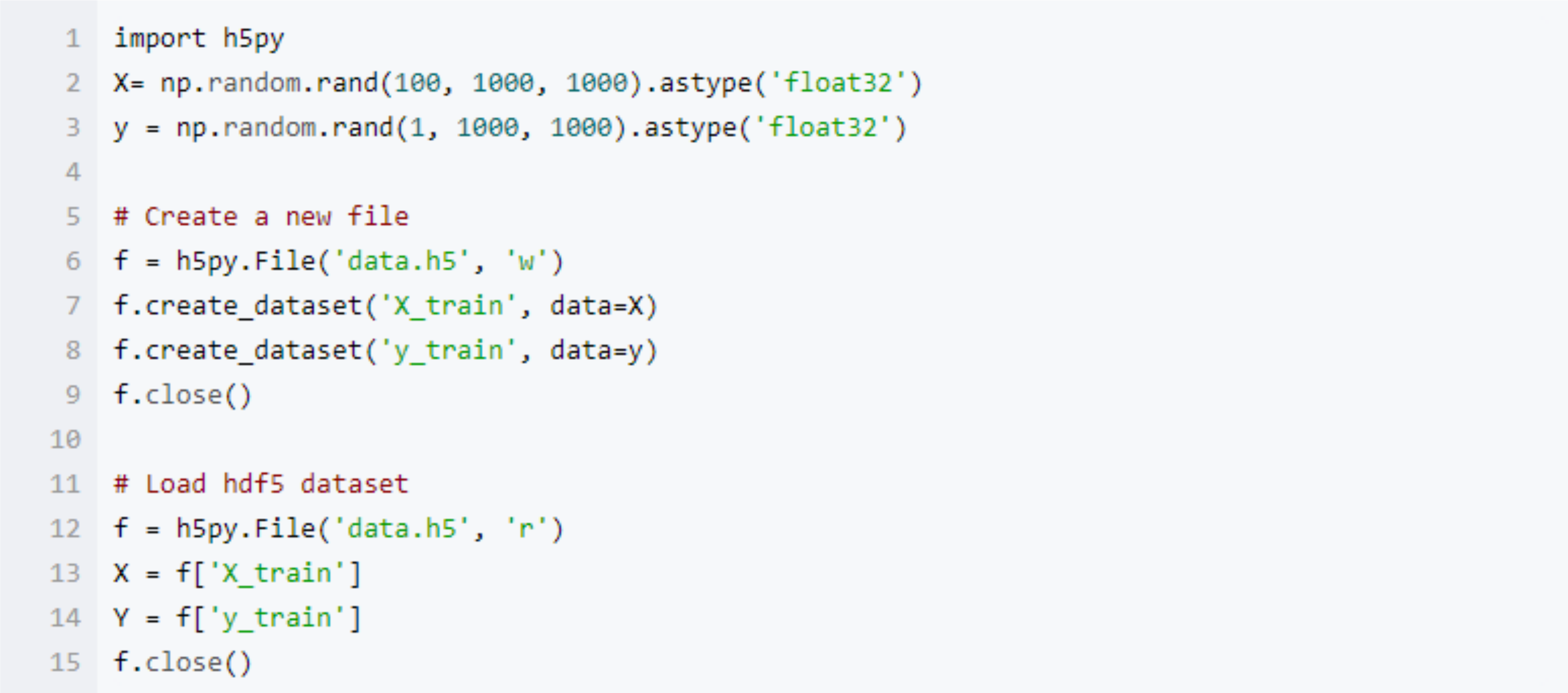
Load H5 文件很快
需要少的内存

数据预处理–创建数据文件 (Python数据存储（压缩））

- numpy.save , numpy.savez , scipy.io.savemat
  - numpy和scipy内建的数据存储方式。
- cPickle + gzip
  - cPickle是pickle内建的数据存储方式，gzip是常用的文件压缩模块。
- h5py
  - h5py是对HDF5文件格式进行读写的python包，关于h5py更多介绍与安装，参考官方网站
  - http://docs.h5py.org/en/latest/quick.html
  - 一个HDF5文件就是一个由两种基本数据对象（groups and datasets）存放多种科学数据的容器：
    - HDF5 dataset: 数据元素的一个多维数组以及支持元数据（metadata）；
    - HDF5 group: 包含0个或多个HDF5对象以及支持元数据（metadata）的一个群组结构；
  - h5py的优势：速度快、压缩效率高，总之，numpy.savez和cPickle存储work或不work的都可以试一试h5py！

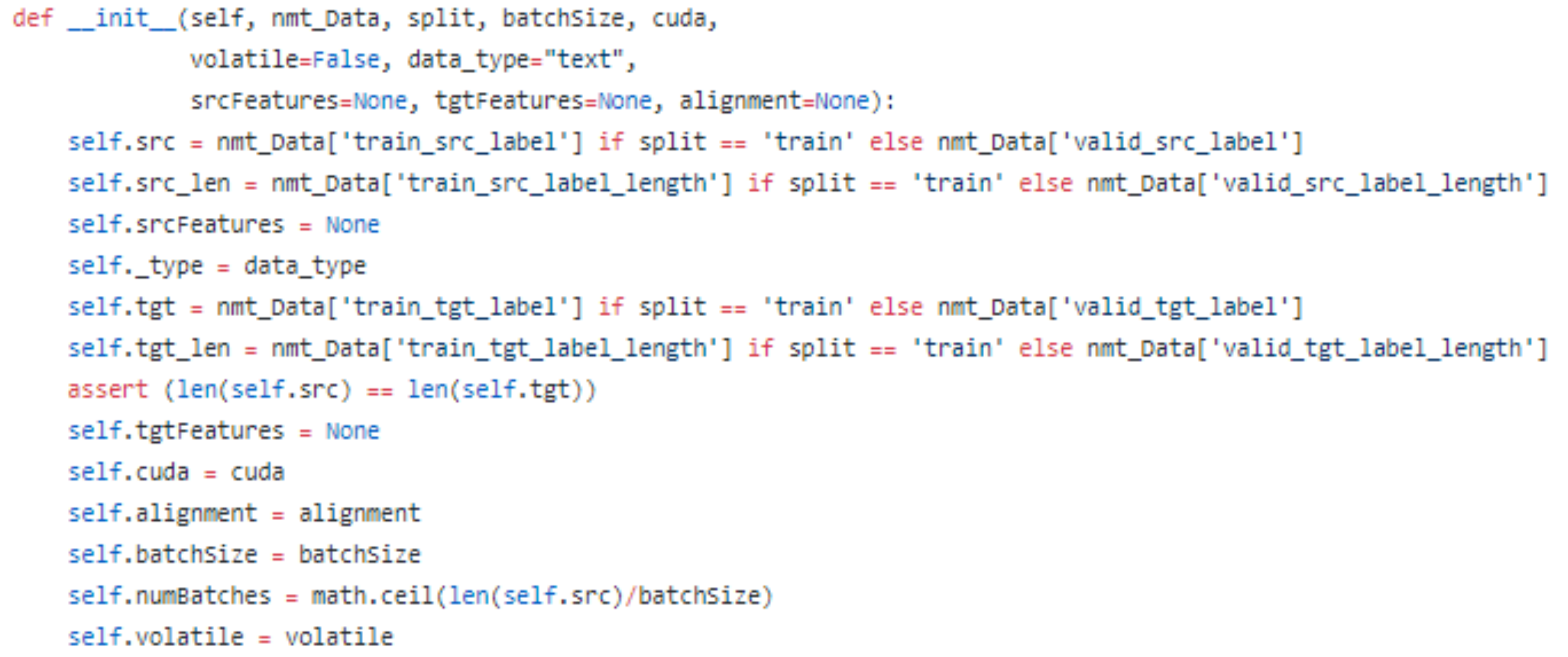
数据预处理—创建数据文件 (Python数据存储（压缩））

❑ h5py读取和存储数据示例

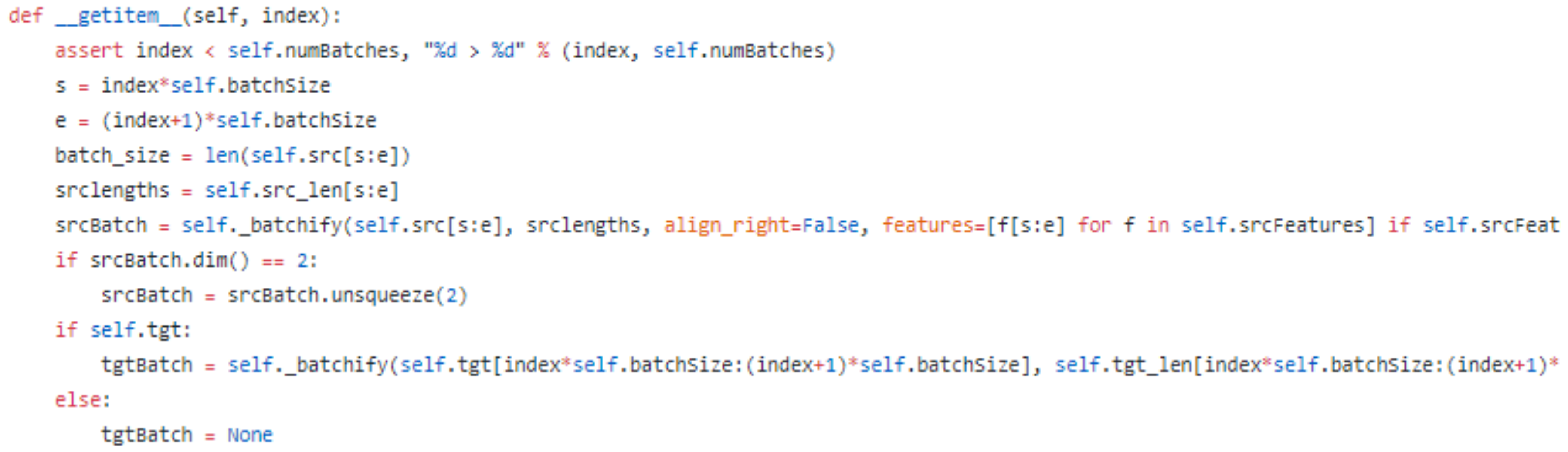


```
import h5py
X= np.random.rand(100, 1000, 1000).astype('float32')
y = np.random.rand(1, 1000, 1000).astype('float32')

# Create a new file
f = h5py.File('data.h5', 'w')
f.create_dataset('X_train', data=X)
f.create_dataset('y_train', data=y)
f.close()

# Load hdf5 dataset
f = h5py.File('data.h5', 'r')
X = f['X_train']
Y = f['y_train']
f.close()
```

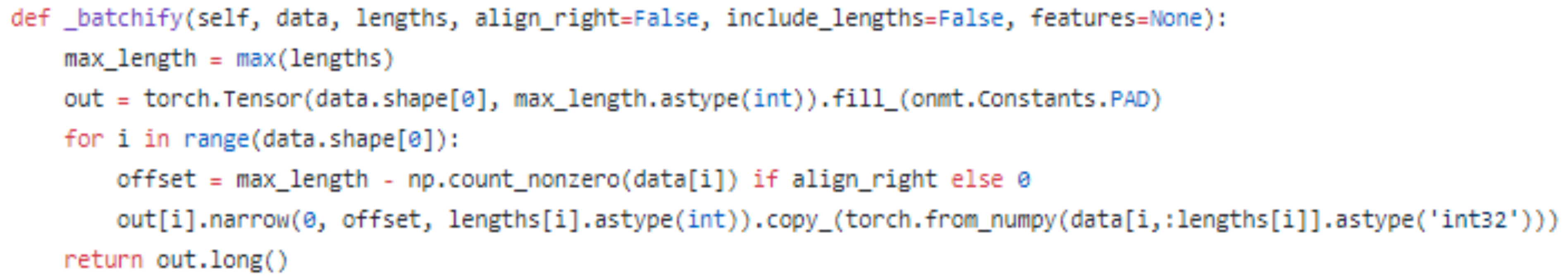
数据预处理—数据加载



```
def __init__(self, nmt_Data, split, batchSize, cuda,
             volatile=False, data_type="text",
             srcFeatures=None, tgtFeatures=None, alignment=None):
    self.src = nmt_Data['train_src_label'] if split == 'train' else nmt_Data['valid_src_label']
    self.src_len = nmt_Data['train_src_label_length'] if split == 'train' else nmt_Data['valid_src_label_length']
    self.srcFeatures = None
    self._type = data_type
    self.tgt = nmt_Data['train_tgt_label'] if split == 'train' else nmt_Data['valid_tgt_label']
    self.tgt_len = nmt_Data['train_tgt_label_length'] if split == 'train' else nmt_Data['valid_tgt_label_length']
    assert (len(self.src) == len(self.tgt))
    self.tgtFeatures = None
    self.cuda = cuda
    self.alignment = alignment
    self.batchSize = batchSize
    self.numBatches = math.ceil(len(self.src)/batchSize)
    self.volatile = volatile
```

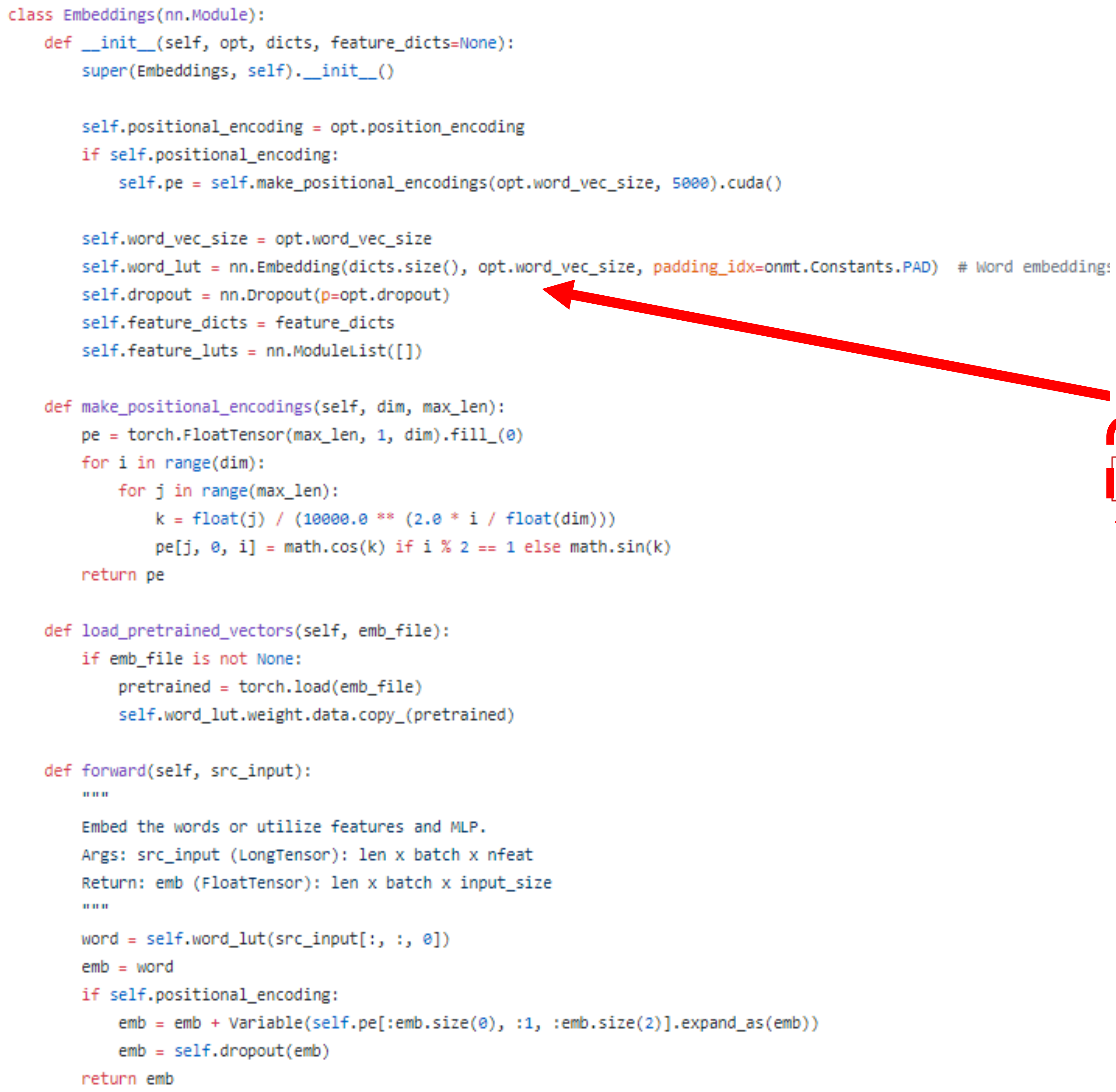


```
def __getitem__(self, index):
    assert index < self.numBatches, "%d > %d" % (index, self.numBatches)
    s = index*self.batchSize
    e = (index+1)*self.batchSize
    batch_size = len(self.src[s:e])
    srclengths = self.src_len[s:e]
    srcBatch = self._batchify(self.src[s:e], srclengths, align_right=False, features=[f[s:e] for f in self.srcFeatures] if self.srcFeat
    if srcBatch.dim() == 2:
        srcBatch = srcBatch.unsqueeze(2)
    if self.tgt:
        tgtBatch = self._batchify(self.tgt[index*self.batchSize:(index+1)*self.batchSize], self.tgt_len[index*self.batchSize:(index+1)*
    else:
        tgtBatch = None
```

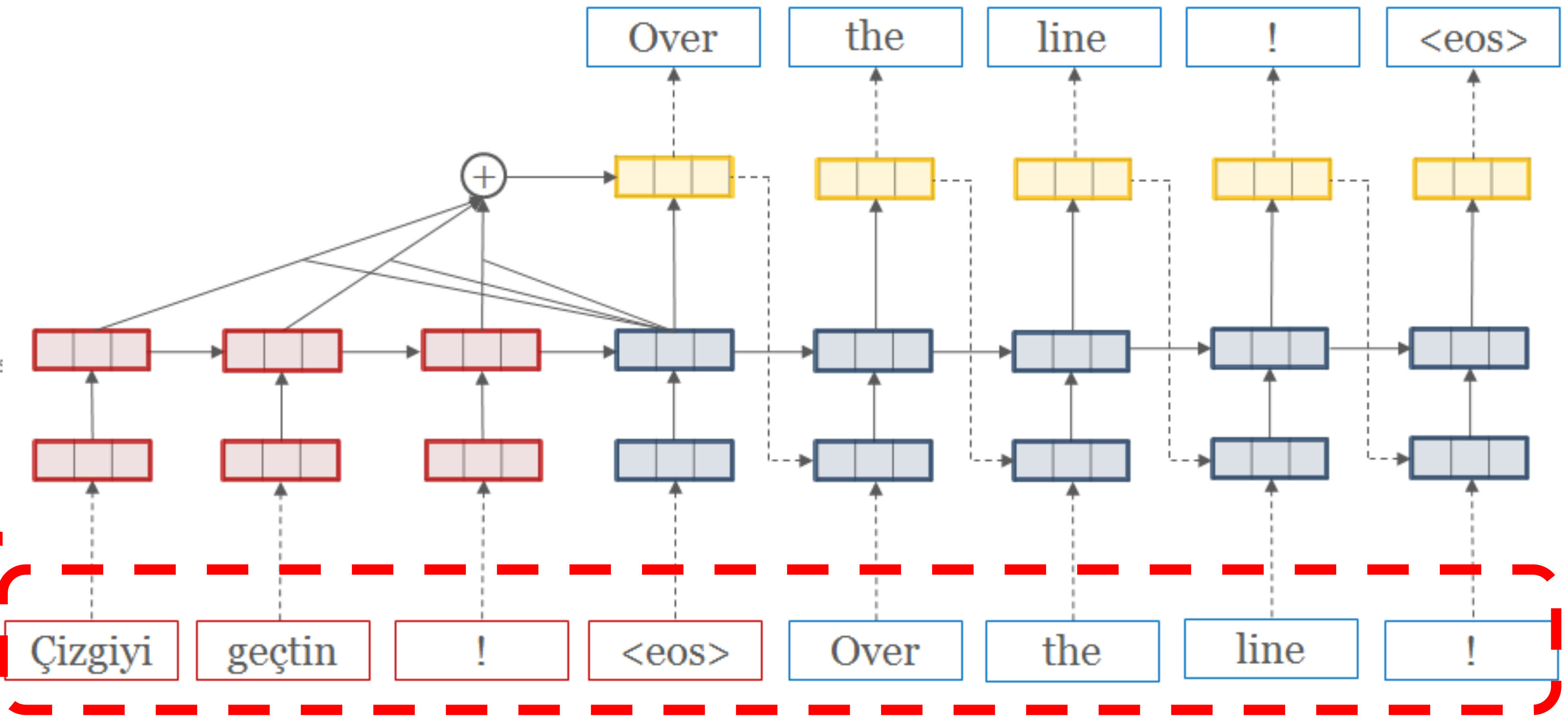


```
def _batchify(self, data, lengths, align_right=False, include_lengths=False, features=None):
    max_length = max(lengths)
    out = torch.Tensor(data.shape[0], max_length.astype(int)).fill_(onmt.Constants.PAD)
    for i in range(data.shape[0]):
        offset = max_length - np.count_nonzero(data[i]) if align_right else 0
        out[i].narrow(0, offset, lengths[i].astype(int)).copy_(torch.from_numpy(data[i,:lengths[i]].astype('int32')))
    return out.long()
```

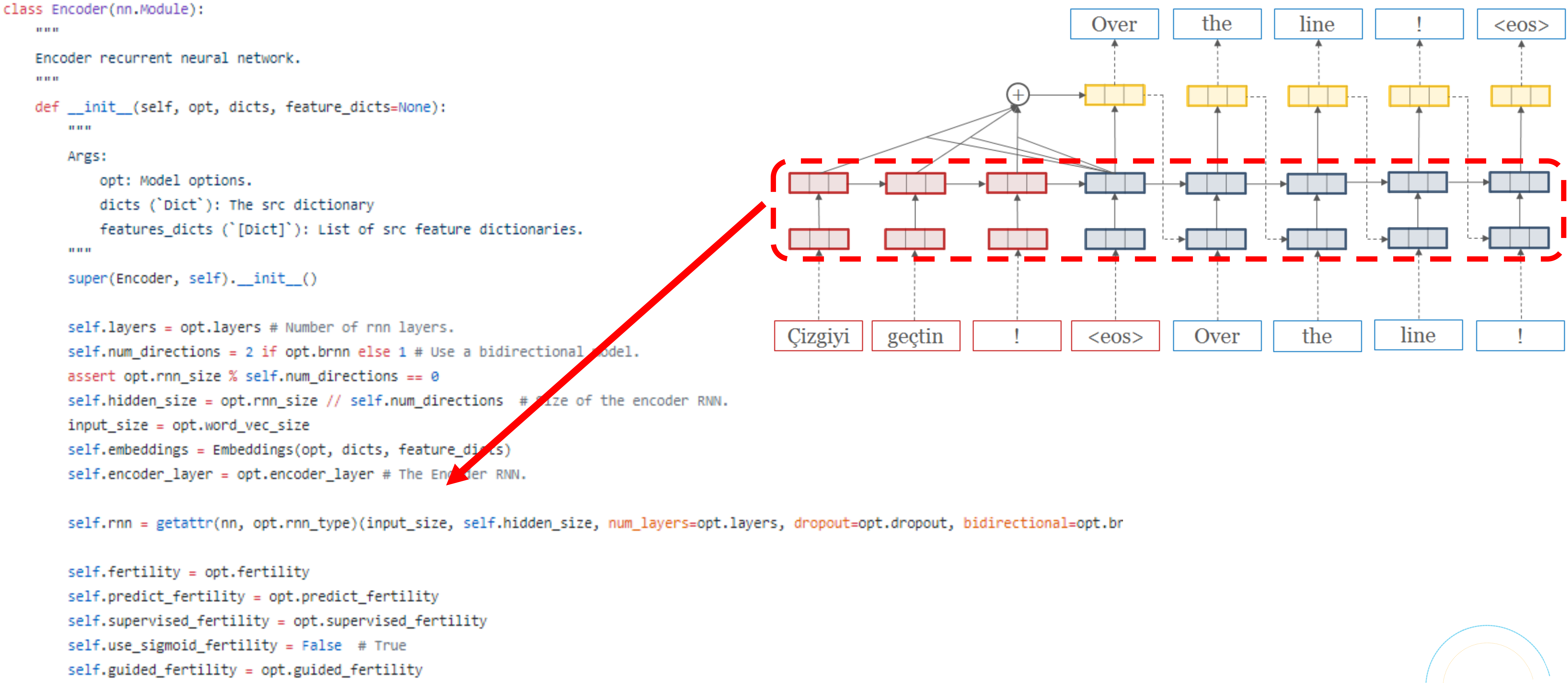
模型–Word Embedding



```
class Embeddings(nn.Module):
    def __init__(self, opt, dicts, feature_dicts=None):
        super(Embeddings, self).__init__()

        self.positional_encoding = opt.position_encoding
        if self.positional_encoding:
            self.pe = self.make_positional_encodings(opt.word_vec_size, 5000).cuda()

        self.word_vec_size = opt.word_vec_size
        self.word_lut = nn.Embedding(dicts.size(), opt.word_vec_size, padding_idx=onmt.Constants.PAD)  # Word embedding
        self.dropout = nn.Dropout(p=opt.dropout)
        self.feature_dicts = feature_dicts
        self.feature_luts = nn.ModuleList([])

    def make_positional_encodings(self, dim, max_len):
        pe = torch.FloatTensor(max_len, 1, dim).fill_(0)
        for i in range(dim):
            for j in range(max_len):
                k = float(j) / (10000.0 ** (2.0 * i / float(dim)))
                pe[j, 0, i] = math.cos(k) if i % 2 == 1 else math.sin(k)
        return pe

    def load_pretrained_vectors(self, emb_file):
        if emb_file is not None:
            pretrained = torch.load(emb_file)
            self.word_lut.weight.data.copy_(pretrained)

    def forward(self, src_input):
        """
        Embed the words or utilize features and MLP.
        Args: src_input (LongTensor): len x batch x nfeat
        Return: emb (FloatTensor): len x batch x input_size
        """
        word = self.word_lut(src_input[:, :, 0])
        emb = word
        if self.positional_encoding:
            emb = emb + Variable(self.pe[:emb.size(0), :1, :emb.size(2)].expand_as(emb))
            emb = self.dropout(emb)
        return emb
```

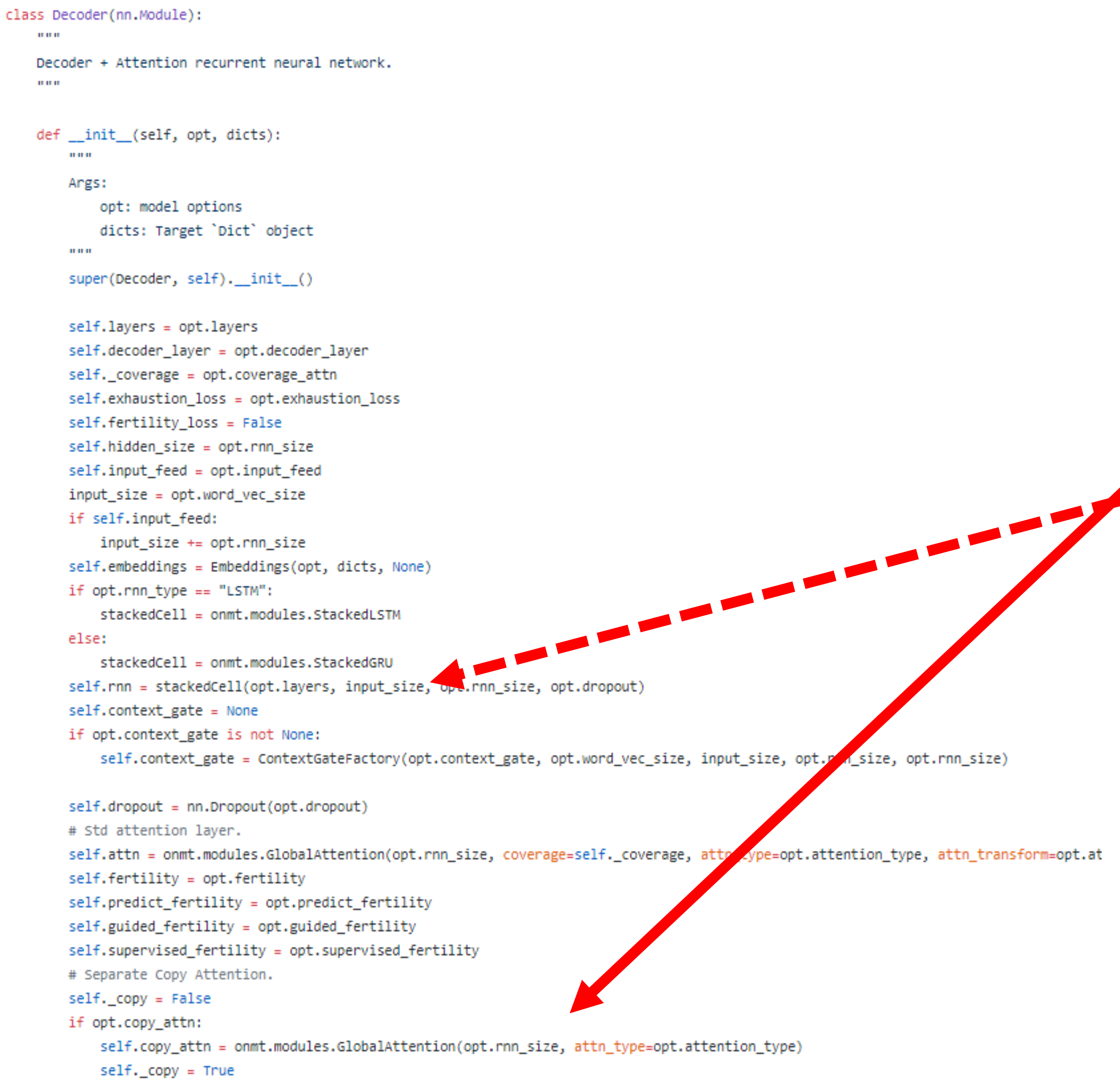


模型—Encoder

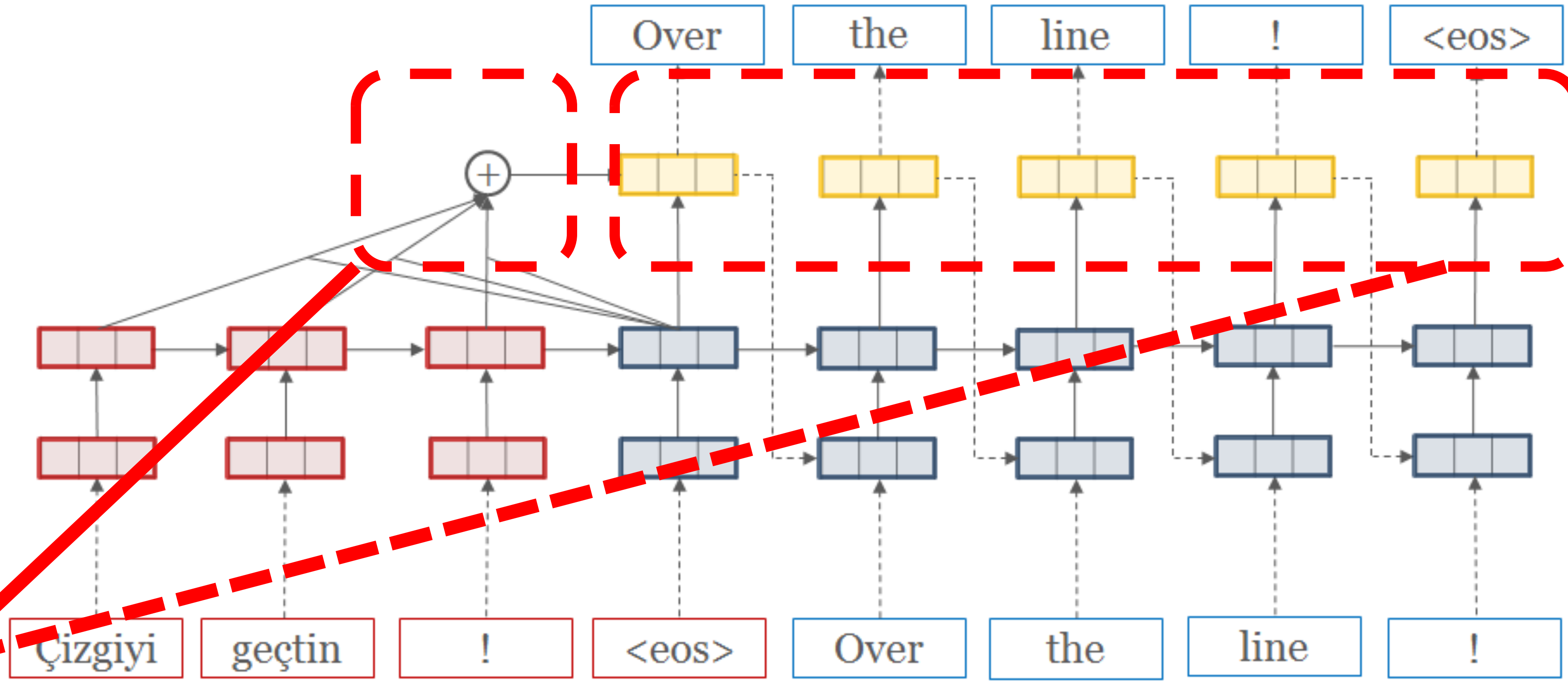


```
class Encoder(nn.Module):
    """
    Encoder recurrent neural network.
    """
    def __init__(self, opt, dicts, feature_dicts=None):
        """
        Args:
            opt: Model options.
            dicts (`Dict`): The src dictionary
            features_dicts (`[Dict]`): List of src feature dictionaries.
        """
        super(Encoder, self).__init__()

        self.layers = opt.layers # Number of rnn layers.
        self.num_directions = 2 if opt.brnn else 1 # Use a bidirectional model.
        assert opt.rnn_size % self.num_directions == 0
        self.hidden_size = opt.rnn_size // self.num_directions  # Size of the encoder RNN.
        input_size = opt.word_vec_size
        self.embeddings = Embeddings(opt, dicts, feature_dicts)
        self.encoder_layer = opt.encoder_layer # The Encoder RNN.

        self.rnn = getattr(nn, opt.rnn_type)(input_size, self.hidden_size, num_layers=opt.layers, dropout=opt.dropout, bidirectional=opt.br

        self.fertility = opt.fertility
        self.predict_fertility = opt.predict_fertility
        self.supervised_fertility = opt.supervised_fertility
        self.use_sigmoid_fertility = False  # True
        self.guided_fertility = opt.guided_fertility
```

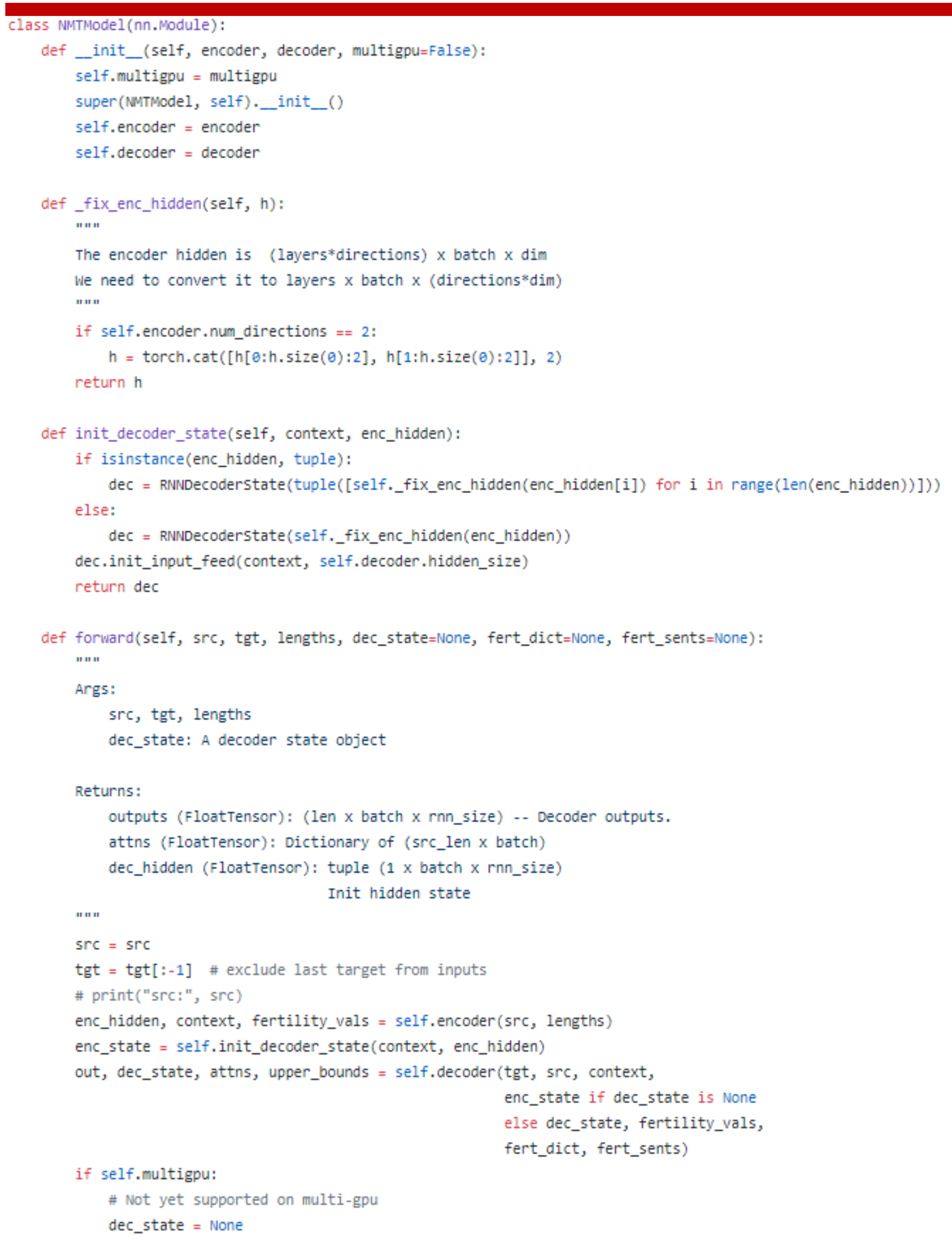
模型—Decoder



```
class Decoder(nn.Module):
    """
    Decoder + Attention recurrent neural network.
    """

    def __init__(self, opt, dicts):
        """
        Args:
            opt: model options
            dicts: Target `Dict` object
        """
        super(Decoder, self).__init__()

        self.layers = opt.layers
        self.decoder_layer = opt.decoder_layer
        self._coverage = opt.coverage_attn
        self.exhaustion_loss = opt.exhaustion_loss
        self.fertility_loss = False
        self.hidden_size = opt.rnn_size
        self.input_feed = opt.input_feed
        input_size = opt.word_vec_size
        if self.input_feed:
            input_size += opt.rnn_size
        self.embeddings = Embeddings(opt, dicts, None)
        if opt.rnn_type == "LSTM":
            stackedCell = onmt.modules.StackedLSTM
        else:
            stackedCell = onmt.modules.StackedGRU
        self.rnn = stackedCell(opt.layers, input_size, opt.rnn_size, opt.dropout)
        self.context_gate = None
        if opt.context_gate is not None:
            self.context_gate = ContextGateFactory(opt.context_gate, opt.word_vec_size, input_size, opt.rnn_size, opt.rnn_size)

        self.dropout = nn.Dropout(opt.dropout)
        # Std attention layer.
        self.attn = onmt.modules.GlobalAttention(opt.rnn_size, coverage=self._coverage, attn_type=opt.attention_type, attn_transform=opt.at
        self.fertility = opt.fertility
        self.predict_fertility = opt.predict_fertility
        self.guided_fertility = opt.guided_fertility
        self.supervised_fertility = opt.supervised_fertility
        # Separate Copy Attention.
        self._copy = False
        if opt.copy_attn:
            self.copy_attn = onmt.modules.GlobalAttention(opt.rnn_size, attn_type=opt.attention_type)
            self._copy = True
```

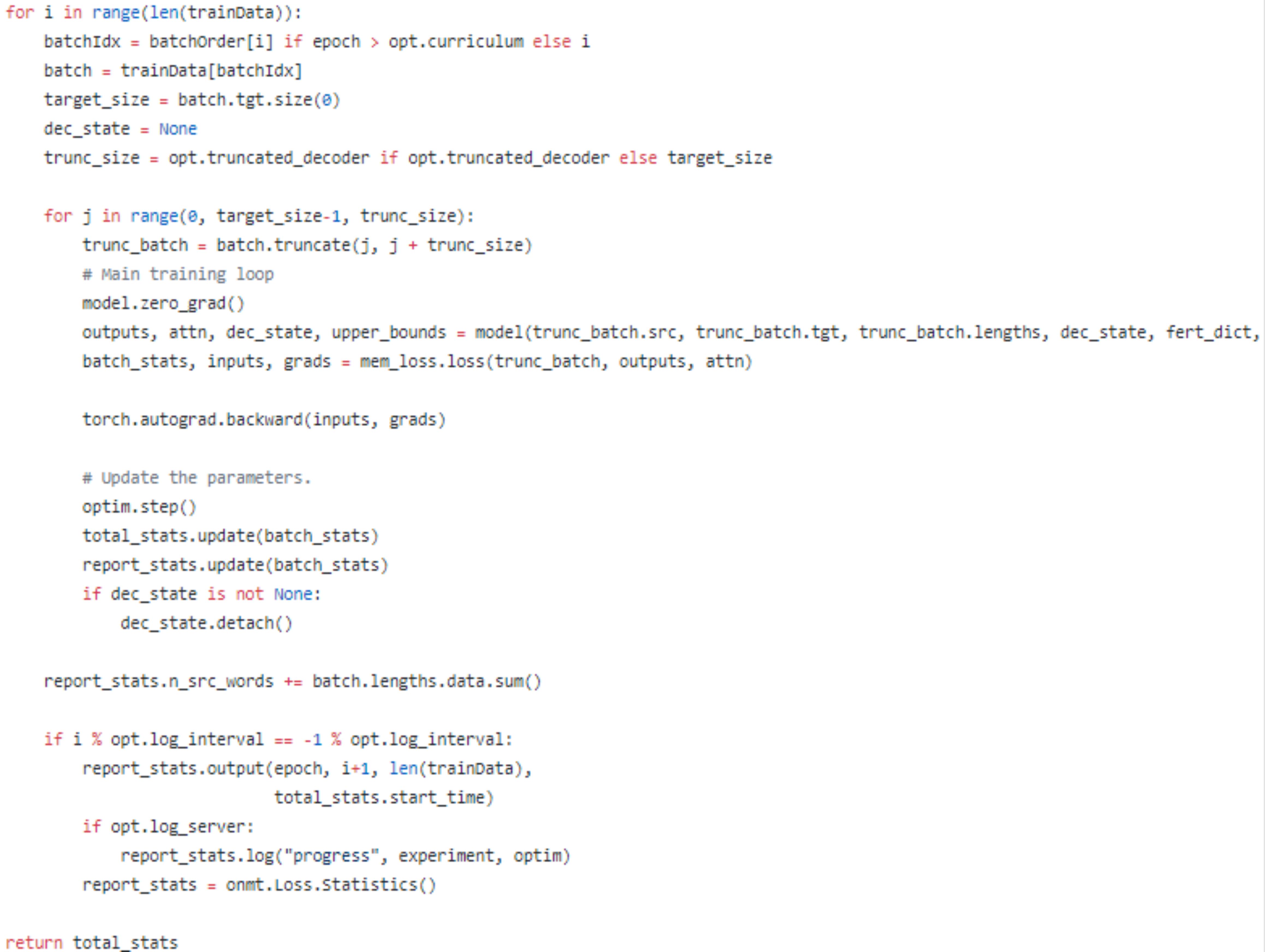


模型—Neural Machine Translation Model



```python
class NMTModel(nn.Module):
    def __init__(self, encoder, decoder, multigpu=False):
        self.multigpu = multigpu
        super(NMTModel, self).__init__()
        self.encoder = encoder
        self.decoder = decoder

    def _fix_enc_hidden(self, h):
        """
        The encoder hidden is  (layers*directions) x batch x dim
        We need to convert it to layers x batch x (directions*dim)
        """
        if self.encoder.num_directions == 2:
            h = torch.cat([h[0:h.size(0):2], h[1:h.size(0):2]], 2)
        return h

    def init_decoder_state(self, context, enc_hidden):
        if isinstance(enc_hidden, tuple):
            dec = RNNDecoderState(tuple([self._fix_enc_hidden(enc_hidden[i]) for i in range(len(enc_hidden))]))
        else:
            dec = RNNDecoderState(self._fix_enc_hidden(enc_hidden))
        dec.init_input_feed(context, self.decoder.hidden_size)
        return dec

    def forward(self, src, tgt, lengths, dec_state=None, fert_dict=None, fert_sents=None):
        """
        Args:
            src, tgt, lengths
            dec_state: A decoder state object

        Returns:
            outputs (FloatTensor): (len x batch x rnn_size) -- Decoder outputs.
            attns (FloatTensor): Dictionary of (src_len x batch)
            dec_hidden (FloatTensor): tuple (1 x batch x rnn_size)
                                      Init hidden state
        """
        src = src
        tgt = tgt[:-1]  # exclude last target from inputs
        # print("src:", src)
        enc_hidden, context, fertility_vals = self.encoder(src, lengths)
        enc_state = self.init_decoder_state(context, enc_hidden)
        out, dec_state, attns, upper_bounds = self.decoder(tgt, src, context,
                                                          enc_state if dec_state is None
                                                          else dec_state, fertility_vals,
                                                          fert_dict, fert_sents)
        if self.multigpu:
            # Not yet supported on multi-gpu
            dec_state = None
```

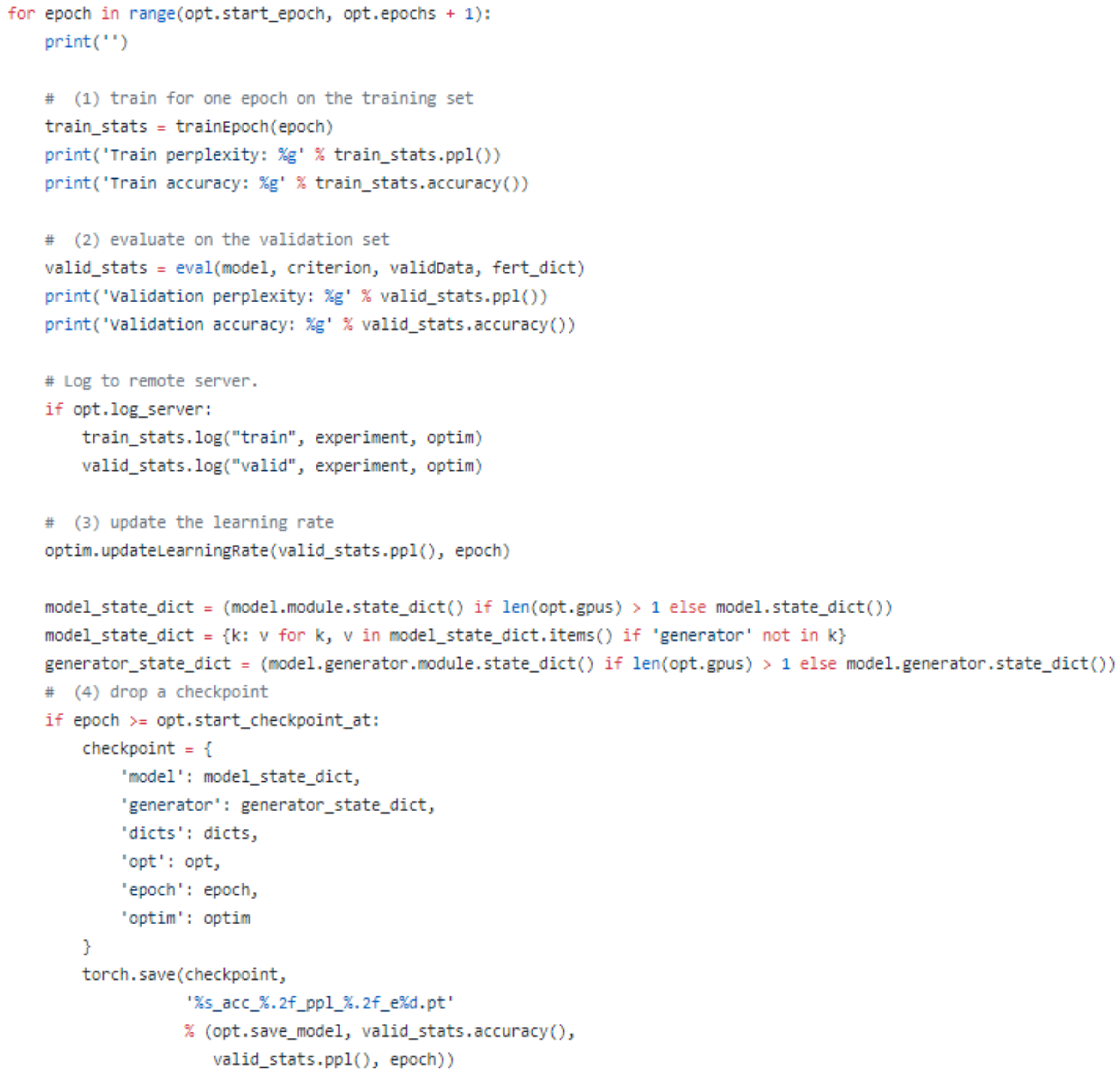
模型—Training



```
for i in range(len(trainData)):
    batchIdx = batchOrder[i] if epoch > opt.curriculum else i
    batch = trainData[batchIdx]
    target_size = batch.tgt.size(0)
    dec_state = None
    trunc_size = opt.truncated_decoder if opt.truncated_decoder else target_size

    for j in range(0, target_size-1, trunc_size):
        trunc_batch = batch.truncate(j, j + trunc_size)
        # Main training loop
        model.zero_grad()
        outputs, attn, dec_state, upper_bounds = model(trunc_batch.src, trunc_batch.tgt, trunc_batch.lengths, dec_state, fert_dict,
        batch_stats, inputs, grads = mem_loss.loss(trunc_batch, outputs, attn)

        torch.autograd.backward(inputs, grads)

        # Update the parameters.
        optim.step()
        total_stats.update(batch_stats)
        report_stats.update(batch_stats)
        if dec_state is not None:
            dec_state.detach()

    report_stats.n_src_words += batch.lengths.data.sum()

    if i % opt.log_interval == -1 % opt.log_interval:
        report_stats.output(epoch, i+1, len(trainData),
                            total_stats.start_time)
        if opt.log_server:
            report_stats.log("progress", experiment, optim)
        report_stats = onmt.Loss.Statistics()

return total_stats
```

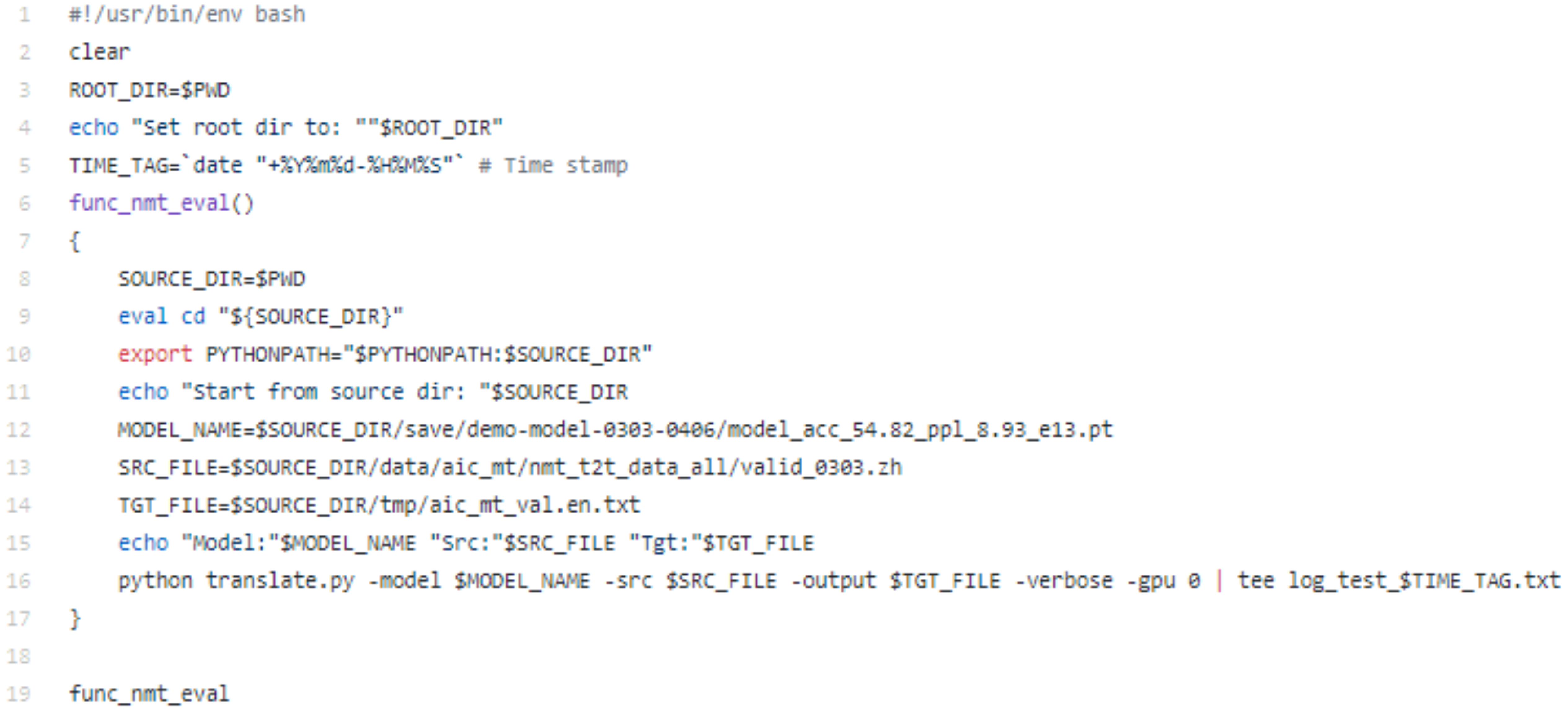
$$CE = -\sum_{x} p(x)\,\log q(x)$$

验证—Evaluation



```
for epoch in range(opt.start_epoch, opt.epochs + 1):
    print('')

    #  (1) train for one epoch on the training set
    train_stats = trainEpoch(epoch)
    print('Train perplexity: %g' % train_stats.ppl())
    print('Train accuracy: %g' % train_stats.accuracy())

    #  (2) evaluate on the validation set
    valid_stats = eval(model, criterion, validData, fert_dict)
    print('Validation perplexity: %g' % valid_stats.ppl())
    print('Validation accuracy: %g' % valid_stats.accuracy())

    # Log to remote server.
    if opt.log_server:
        train_stats.log("train", experiment, optim)
        valid_stats.log("valid", experiment, optim)

    #  (3) update the learning rate
    optim.updateLearningRate(valid_stats.ppl(), epoch)

    model_state_dict = (model.module.state_dict() if len(opt.gpus) > 1 else model.state_dict())
    model_state_dict = {k: v for k, v in model_state_dict.items() if 'generator' not in k}
    generator_state_dict = (model.generator.module.state_dict() if len(opt.gpus) > 1 else model.generator.state_dict())
    #  (4) drop a checkpoint
    if epoch >= opt.start_checkpoint_at:
        checkpoint = {
            'model': model_state_dict,
            'generator': generator_state_dict,
            'dicts': dicts,
            'opt': opt,
            'epoch': epoch,
            'optim': optim
        }
        torch.save(checkpoint,
                   '%s_acc_%.2f_ppl_%.2f_e%d.pt'
                   % (opt.save_model, valid_stats.accuracy(),
                      valid_stats.ppl(), epoch))
```

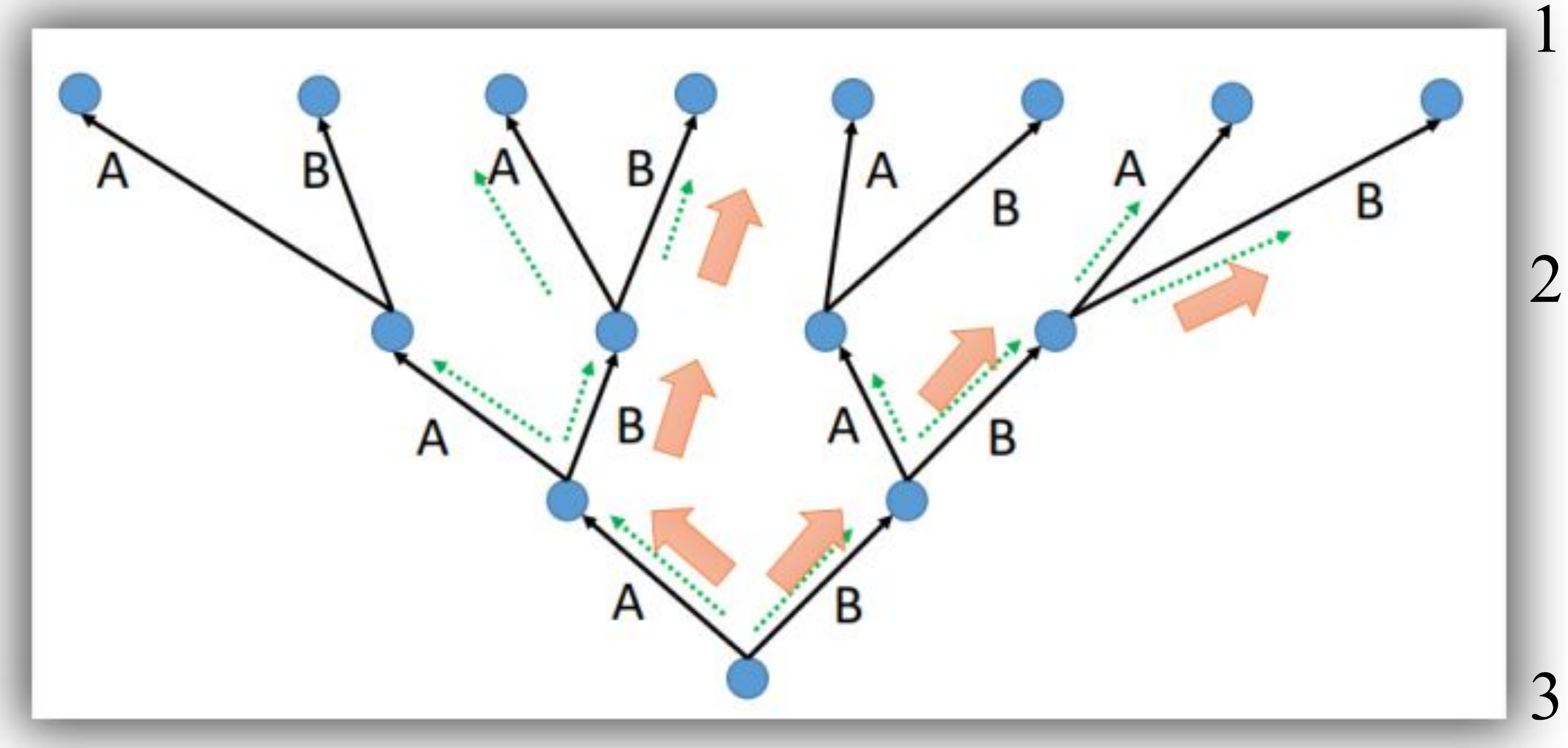
验证—Testing



```
#!/usr/bin/env bash
clear
ROOT_DIR=$PWD
echo "Set root dir to: ""$ROOT_DIR"
TIME_TAG=`date "+%Y%m%d-%H%M%S"` # Time stamp
func_nmt_eval()
{
    SOURCE_DIR=$PWD
    eval cd "${SOURCE_DIR}"
    export PYTHONPATH="$PYTHONPATH:$SOURCE_DIR"
    echo "Start from source dir: "$SOURCE_DIR
    MODEL_NAME=$SOURCE_DIR/save/demo-model-0303-0406/model_acc_54.82_ppl_8.93_e13.pt
    SRC_FILE=$SOURCE_DIR/data/aic_mt/nmt_t2t_data_all/valid_0303.zh
    TGT_FILE=$SOURCE_DIR/tmp/aic_mt_val.en.txt
    echo "Model:"$MODEL_NAME "Src:"$SRC_FILE "Tgt:"$TGT_FILE
    python translate.py -model $MODEL_NAME -src $SRC_FILE -output $TGT_FILE -verbose -gpu 0 | tee log_test_$TIME_TAG.txt
}

func_nmt_eval
```

验证—TESTING （beam search）

Beam search是一个搜索策略，对于语言生成的模型中，给定语言模型，它可以搜索出更差异化、更合理的结果。beam search功能上等价于最简单的单步最大概率，或者viterbi算法等。

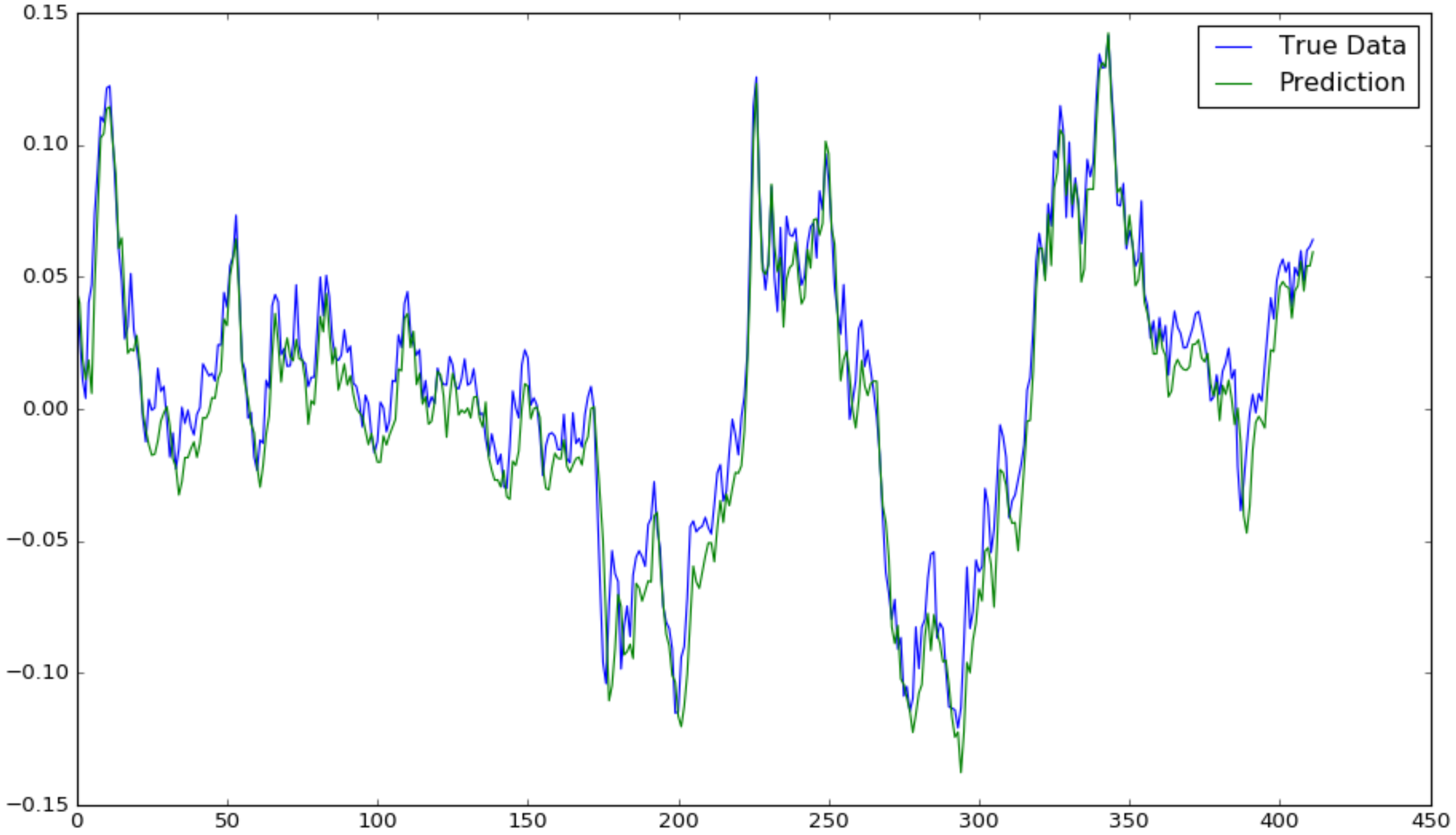


这里我们假设Beam_size=2. 意思是每个`word`后面的分支考虑概率最大的那两个`words`。

1. 比如左边的例子，从下往上首先分成A、B两个words，然后继续往上传播，句子变成是AA/AB/BA/BB这四种情况（绿色虚线）。
2. 考虑到`Beam Size=2`，选择概率最大的两个，假设是AB/BA（橙色大箭头）。然后以选择的AB/BA继续向上传播，又出现了四种情况ABA/ABB/BBA/BBB，依然是选择综合概率最大的两个ABB/BBB。
3. 以此类推，直至句子结束。只要可以调整好`Beam Size`，就能够使用最小的计算量，得到最优的结果。

Seq2seq实践—股票预测

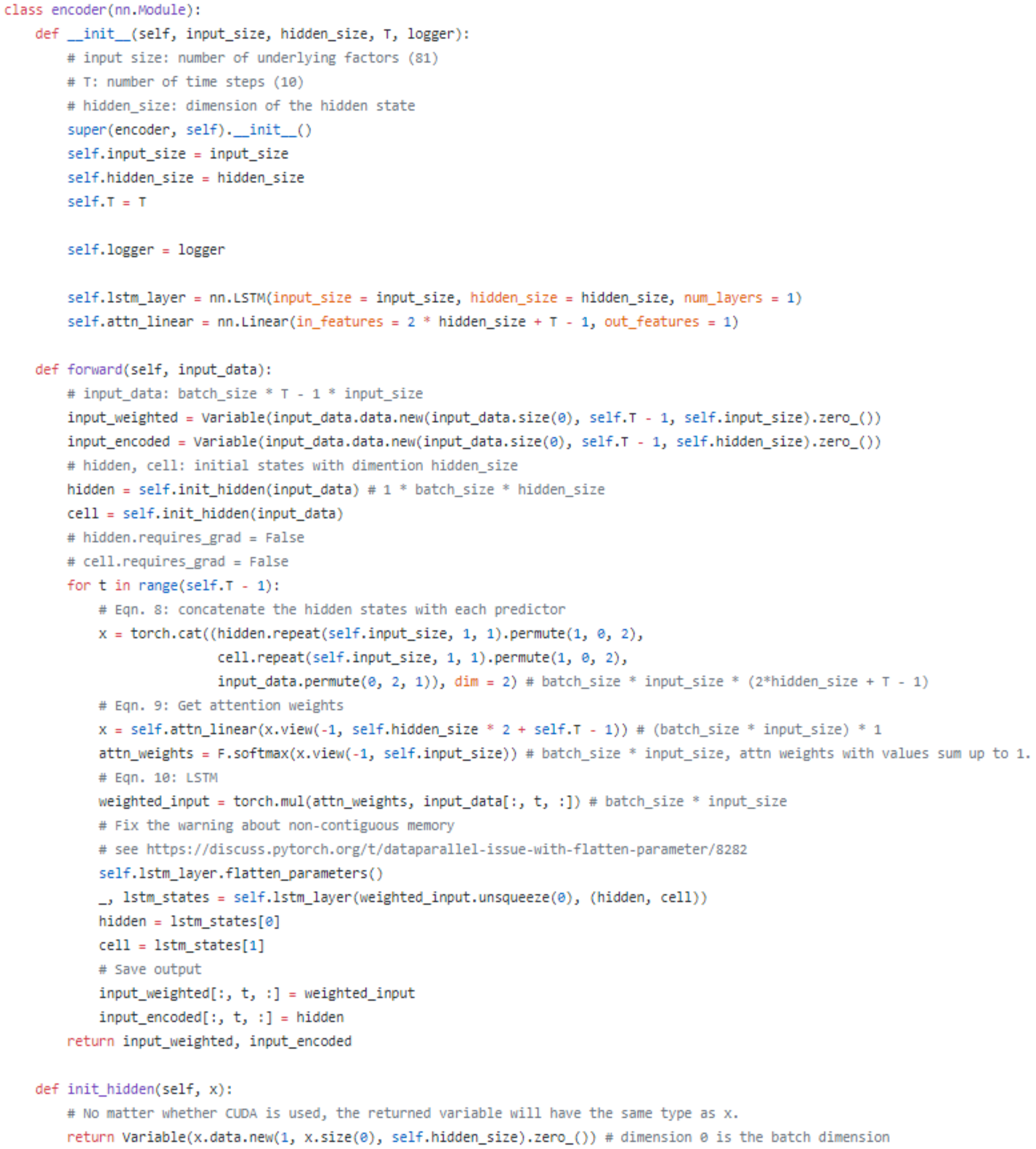
STOCK预测



http://cseweb.ucsd.edu/~yaq007/NASDAQ100_stock_data.html

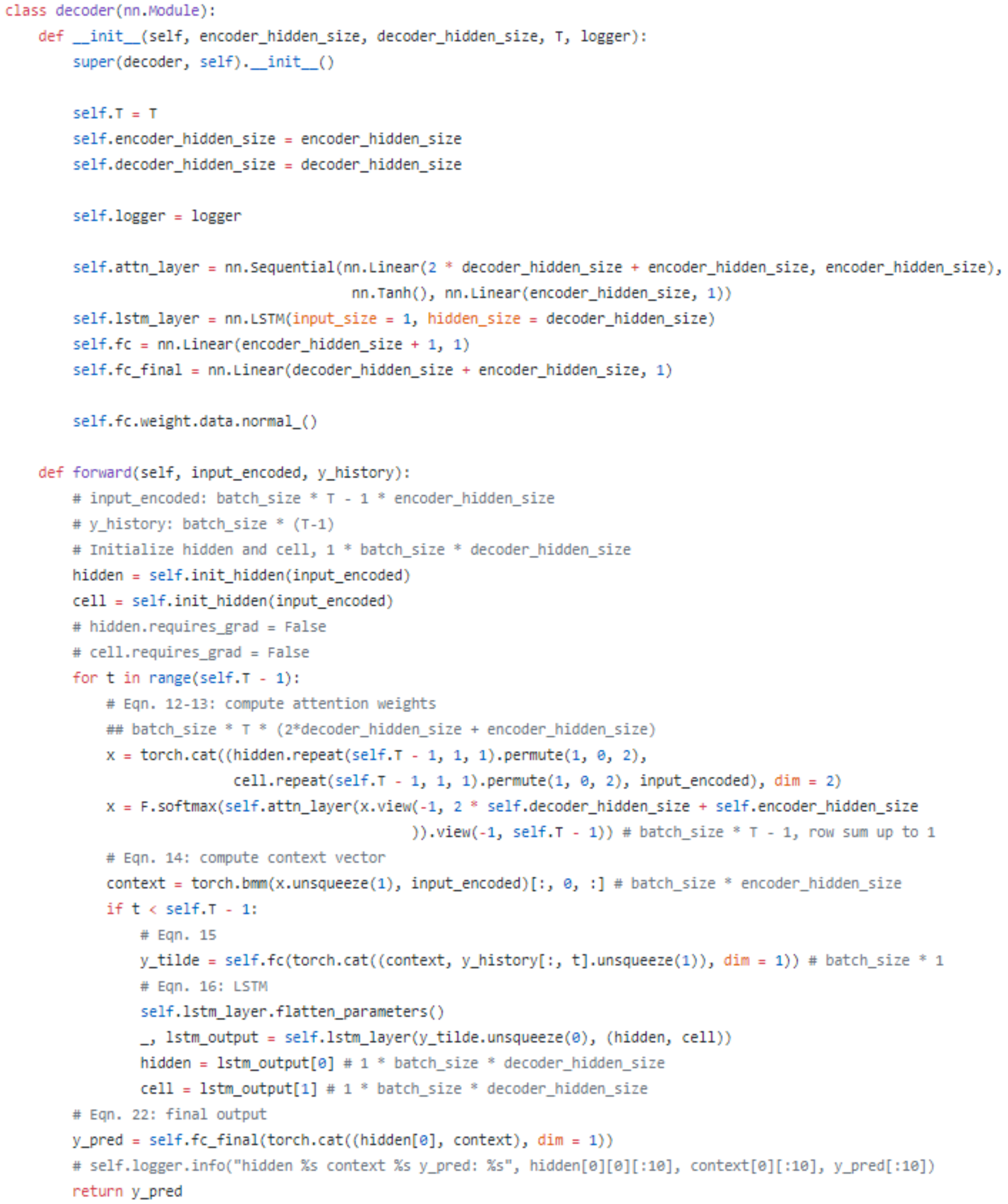
NASDAQ 100 stock dataset consists of stock prices of 104 corporations under NASDAQ 100 and the index value of NASDAQ 100. The frequency of the data collection is one-minute. This data covers the period from July 26, 2016 to April 28, 2017, in total 191 days.

Stock : Encoder



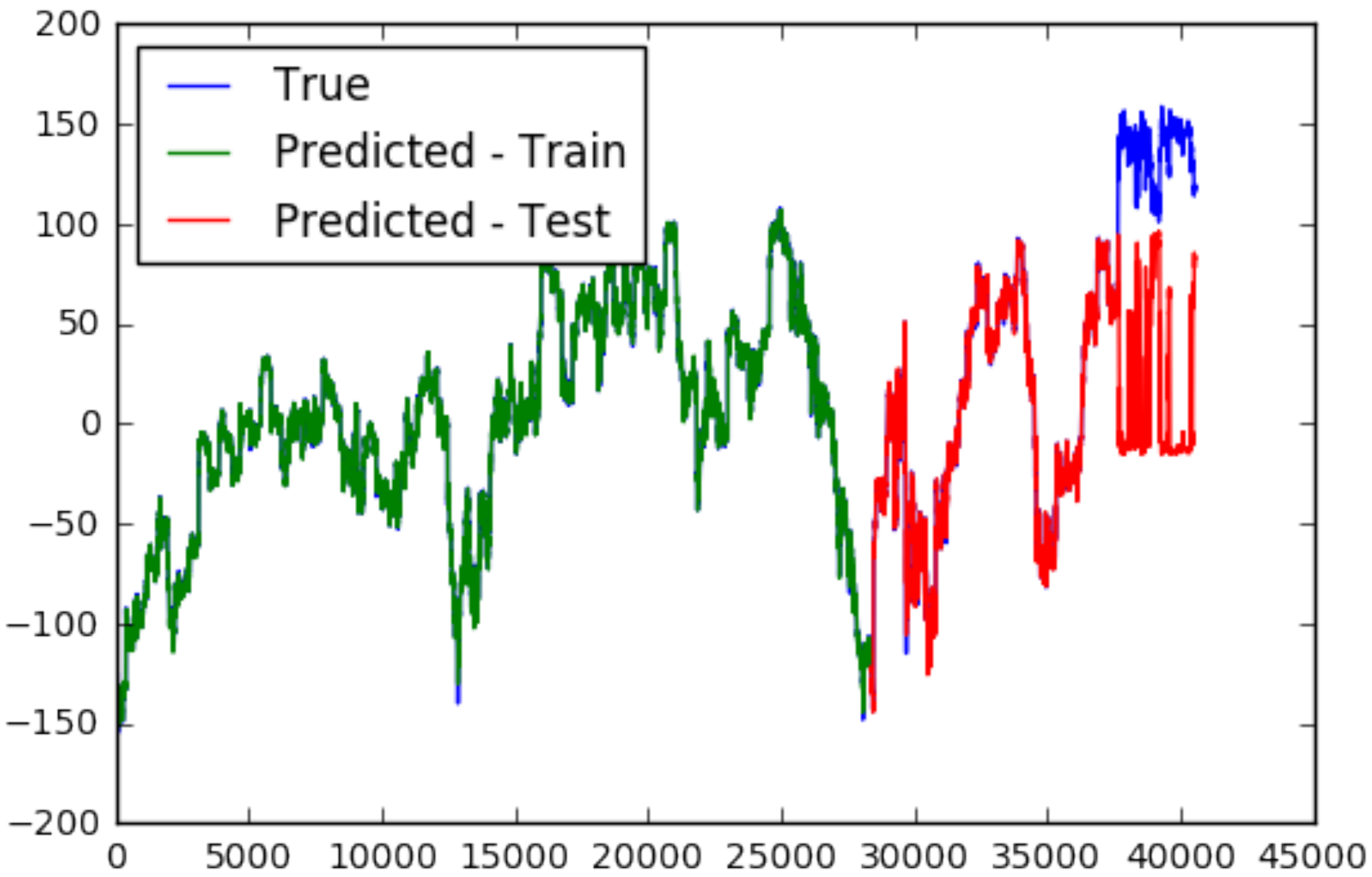
```python
class encoder(nn.Module):
    def __init__(self, input_size, hidden_size, T, logger):
        # input size: number of underlying factors (81)
        # T: number of time steps (10)
        # hidden_size: dimension of the hidden state
        super(encoder, self).__init__()
        self.input_size = input_size
        self.hidden_size = hidden_size
        self.T = T

        self.logger = logger

        self.lstm_layer = nn.LSTM(input_size = input_size, hidden_size = hidden_size, num_layers = 1)
        self.attn_linear = nn.Linear(in_features = 2 * hidden_size + T - 1, out_features = 1)

    def forward(self, input_data):
        # input_data: batch_size * T - 1 * input_size
        input_weighted = Variable(input_data.data.new(input_data.size(0), self.T - 1, self.input_size).zero_())
        input_encoded = Variable(input_data.data.new(input_data.size(0), self.T - 1, self.hidden_size).zero_())
        # hidden, cell: initial states with dimention hidden_size
        hidden = self.init_hidden(input_data) # 1 * batch_size * hidden_size
        cell = self.init_hidden(input_data)
        # hidden.requires_grad = False
        # cell.requires_grad = False
        for t in range(self.T - 1):
            # Eqn. 8: concatenate the hidden states with each predictor
            x = torch.cat((hidden.repeat(self.input_size, 1, 1).permute(1, 0, 2),
                           cell.repeat(self.input_size, 1, 1).permute(1, 0, 2),
                           input_data.permute(0, 2, 1)), dim = 2) # batch_size * input_size * (2*hidden_size + T - 1)
            # Eqn. 9: Get attention weights
            x = self.attn_linear(x.view(-1, self.hidden_size * 2 + self.T - 1)) # (batch_size * input_size) * 1
            attn_weights = F.softmax(x.view(-1, self.input_size)) # batch_size * input_size, attn weights with values sum up to 1.
            # Eqn. 10: LSTM
            weighted_input = torch.mul(attn_weights, input_data[:, t, :]) # batch_size * input_size
            # Fix the warning about non-contiguous memory
            # see https://discuss.pytorch.org/t/dataparallel-issue-with-flatten-parameter/8282
            self.lstm_layer.flatten_parameters()
            _, lstm_states = self.lstm_layer(weighted_input.unsqueeze(0), (hidden, cell))
            hidden = lstm_states[0]
            cell = lstm_states[1]
            # Save output
            input_weighted[:, t, :] = weighted_input
            input_encoded[:, t, :] = hidden
        return input_weighted, input_encoded

    def init_hidden(self, x):
        # No matter whether CUDA is used, the returned variable will have the same type as x.
        return Variable(x.data.new(1, x.size(0), self.hidden_size).zero_()) # dimension 0 is the batch dimension
```

Stock : Encoder



```
class decoder(nn.Module):
    def __init__(self, encoder_hidden_size, decoder_hidden_size, T, logger):
        super(decoder, self).__init__()

        self.T = T
        self.encoder_hidden_size = encoder_hidden_size
        self.decoder_hidden_size = decoder_hidden_size

        self.logger = logger

        self.attn_layer = nn.Sequential(nn.Linear(2 * decoder_hidden_size + encoder_hidden_size, encoder_hidden_size),
                                         nn.Tanh(), nn.Linear(encoder_hidden_size, 1))
        self.lstm_layer = nn.LSTM(input_size = 1, hidden_size = decoder_hidden_size)
        self.fc = nn.Linear(encoder_hidden_size + 1, 1)
        self.fc_final = nn.Linear(decoder_hidden_size + encoder_hidden_size, 1)

        self.fc.weight.data.normal_()

    def forward(self, input_encoded, y_history):
        # input_encoded: batch_size * T - 1 * encoder_hidden_size
        # y_history: batch_size * (T-1)
        # Initialize hidden and cell, 1 * batch_size * decoder_hidden_size
        hidden = self.init_hidden(input_encoded)
        cell = self.init_hidden(input_encoded)
        # hidden.requires_grad = False
        # cell.requires_grad = False
        for t in range(self.T - 1):
            # Eqn. 12-13: compute attention weights
            ## batch_size * T * (2*decoder_hidden_size + encoder_hidden_size)
            x = torch.cat((hidden.repeat(self.T - 1, 1, 1).permute(1, 0, 2),
                           cell.repeat(self.T - 1, 1, 1).permute(1, 0, 2), input_encoded), dim = 2)
            x = F.softmax(self.attn_layer(x.view(-1, 2 * self.decoder_hidden_size + self.encoder_hidden_size
                                                )).view(-1, self.T - 1)) # batch_size * T - 1, row sum up to 1
            # Eqn. 14: compute context vector
            context = torch.bmm(x.unsqueeze(1), input_encoded)[:, 0, :] # batch_size * encoder_hidden_size
            if t < self.T - 1:
                # Eqn. 15
                y_tilde = self.fc(torch.cat((context, y_history[:, t].unsqueeze(1)), dim = 1)) # batch_size * 1
                # Eqn. 16: LSTM
                self.lstm_layer.flatten_parameters()
                _, lstm_output = self.lstm_layer(y_tilde.unsqueeze(0), (hidden, cell))
                hidden = lstm_output[0] # 1 * batch_size * decoder_hidden_size
                cell = lstm_output[1] # 1 * batch_size * decoder_hidden_size
        # Eqn. 22: final output
        y_pred = self.fc_final(torch.cat((hidden[0], context), dim = 1))
        # self.logger.info("hidden %s context %s y_pred: %s", hidden[0][0][:10], context[0][:10], y_pred[:10])
        return y_pred
```

Stock : Testing



END