RNNG Code User Guide

(CSLT-TRP-160007)

张诗悦（Shiyue Zhang）
byryuer@gmail.com

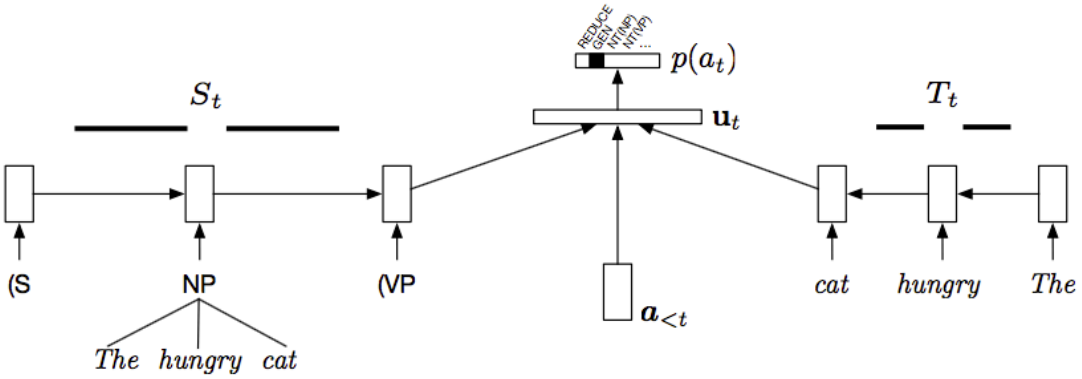
CSLT, RIIT, Tsinghua Univ.
NLP Group

2016/11/26

Table of Contents

1. Introduction

RNNG，即“Recurrent Neural Network Grammars”，是 CMU 大学 Chris Dyer 等人在 NAACL 2016 上发表的论文[1]。该模型是当下做 Parsing 任务中效果最好的模型之一。RNNG 包含两个模型：discriminative model 和 generative model，其中生成模型的效果更好，其模型结构如下图所示。



图一：RNNG Generative Model

这篇文档的主要内容是介绍 RNNG 代码的使用。RNNG 的源代码开源在 Github 上，代码库地址为：https://github.com/clab/rnng。该代码库中也有使用方法的相关介绍，而这篇文档将更加详细地介绍具体的操作步骤，以及对应的参数配置。

特别说明：由于该文档的初衷是为了服务 CSLT 实验室内部使用，因此具体的操作步骤可能存在一定局限性，读者需根据自己的服务器配置做出适应性的调整。对与实验室内部的读者，一些步骤可以省略，直接在/work4 /zhangsy/rnng 目录下拷贝生成好的文件即可。因为纯属个人经验，错误和缺失的地方，还请批评指正。

2. Prepare Code

这部分将介绍在正确运行 RNNG 之前的一些代码准备，包括配置一些依赖库和以及编译代码。

2.1 Boost

RNNG 代码中依赖 C++的 Boost 库，需要在本地配置一个 Boost 库。配置步骤如下：

1. 从 https://sourceforge.net/projects/boost/files/boost/ 上下载 boost 库，版本选择应该没有太大影响，我采用的是 1.61。将下载得到的压缩包，放到自己的根目录，解压。

2. 编译 b2: 执行 *./bootstrap --prefix=[要安装到的目录，默认为当前目录，例如：/work4/zhangsy/boost_1_61_0]* 。

3. 编译 boost: 执行 *./b2* (或者执行./*bjam*)

4. 测试是否配置成功: 将下面这段代码写入 test.cc 文件，执行命令 *g++ -o test test.cc -I [boost 的安装目录，例如: /work4/zhangsy/boost_1_61_0] -L [boost lib 目录，例如: /work4/ zhangsy/boost_1_61_0/stage/lib]*。如果成功运行输出，则说明配置成功。

```
#include <iostream>
#include <boost/timer.hpp>
```

```
using namespace std;
int main()
{
   boost::timer t;
   cout << "max timespan:"<<t.elapsed_max()/3600<<"h"<<endl;
   cout << "min tmiespan:"<<t.elapsed_min()<<"s"<<endl;
   cout<<"now time elapsed:"<<t.elapsed()<<"s"<<endl;
   return 0;
}
```

2.2 Eigen

从 https://bitbucket.org/eigen/eigen/downloads 上下载最新版本的 eigen, 无需安装，只需要把解压后的文件夹，放在根目录(e.g. /work4/zhangsy/)下即可。

2.3 RNNG

这一部分将介绍如何编译 RNNG 的代码，有三种编译方式：普通编译，基于 MKL 编译，基于 CUDA 编译。但是由于 RNNG 的实现决定了其无法利用上 GPU 进行提速，因此我只介绍前两种编译方式。

2.3.1 普通编译

1. 下载 RNNG 代码库： git clone https://github.com/clab/rnng.git
2. 修改 CMakeLists.txt: 在原始的 CMakeLists.txt 中添加下面四句话，其中对应的路径换成读者设置的 Boost 和 eigen 的目录路径即可。

```
SET (BOOST_ROOT "/work4/zhangsy/boost_1_61_0")
SET (Boost_INCLUDE_DIR "/work4/zhangsy/boost_1_61_0")
SET (Boost_LIBRARIES "/work4/zhangsy/boost_1_61_0/stage/lib")
SET (EIGEN3_INCLUDE_DIR "/work4/zhangsy/eigen")
```

3. 执行下面的命令，其中make –j后面的数字是编译时使用的CPU的核数，可以设置为其他值：

```
mkdir build
cd build
cmake ..
make –j 2
```

4. 在rnng下执行命令：*./build/nt-parser/nt-parser –h*，如果正确输出模型的配置参数说明，则说明编译成功。

2.3.2 基于 MKL 编译

1. 同普通编译
2. 同普通编译
3. 在rnng/和cnn/下的CMakeLists.txt中均添加如下：

```
function(find_mkl)
  set(MKL_ARCH intel64)
  find_path(MKL_INCLUDE_DIR mkl.h
            PATHS ${MKL_ROOT} ${MKL_ROOT}/include)
  find_library(MKL_CORE_LIB NAMES mkl_intel_lp64 mkl_intel_thread mkl_core
               PATHS ${MKL_ROOT} ${MKL_ROOT}/lib/${MKL_ARCH}
               DOC "MKL core library path")

  find_library(MKL_COMPILER_LIB NAMES iomp5 libiomp5md
               PATHS ${MKL_ROOT} ${MKL_ROOT}/../compiler/lib/${MKL_ARCH}
#Windows
```

```
${MKL_ROOT}/../compilers_and_libraries/linux/lib/${MKL_ARCH}_lin #Linux
               DOC "MKL compiler lib (for threaded MKL)")

  if(MKL_INCLUDE_DIR AND MKL_CORE_LIB AND MKL_COMPILER_LIB)
    get_filename_component(MKL_CORE_LIB_DIR ${MKL_CORE_LIB} DIRECTORY)
    get_filename_component(MKL_COMPILER_LIB_DIR ${MKL_COMPILER_LIB} DIRECTORY)
    get_filename_component(MKL_COMPILER_LIB_FILE ${MKL_COMPILER_LIB} NAME)
    message(STATUS "Found MKL\n   * include: ${MKL_INCLUDE_DIR},\n   * core library
dir: ${MKL_CORE_LIB_DIR},\n   * compiler library: ${MKL_COMPILER_LIB}")

    # Due to a conflict with /MT and /MD, MSVC needs mkl_intel_lp64 linked last, or
we can change individual
    # projects to use /MT (mkl_intel_lp64 linked with /MT, default MSVC projects
use /MD), or we can instead
    # link to the DLL versions. For now I'm opting for this solution which seems to
work with projects still
    # at their default /MD. Linux build requires the mkl_intel_lp64 to be linked
first. So...:
    if(MSVC)
      set(LIBS ${LIBS} mkl_intel_thread mkl_core mkl_intel_lp64
${MKL_COMPILER_LIB_FILE} PARENT_SCOPE)
    else()
      set(LIBS ${LIBS} mkl_intel_lp64 mkl_intel_thread mkl_core
${MKL_COMPILER_LIB_FILE} PARENT_SCOPE)
    endif()
    include_directories(${MKL_INCLUDE_DIR})
    link_directories(${MKL_CORE_LIB_DIR} ${MKL_COMPILER_LIB_DIR})
    set(MKL_LINK_DIRS ${MKL_CORE_LIB_DIR} ${MKL_COMPILER_LIB_DIR} PARENT_SCOPE) #
Keeping this for python build
  else()
    message(FATAL_ERROR "Failed to find MKL in path: ${MKL_ROOT} (Did you set
MKL_ROOT properly?)")
  endif()
endfunction()

######## Cross-compiler, cross-platform options
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -DEIGEN_FAST_MATH")
if (MKL OR MKL_ROOT)
  find_mkl()  # sets include/lib directories and sets ${LIBS} needed for linking
  set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -DEIGEN_USE_MKL_ALL")
endif()
```

可以参见/work4/zhangsy/rnng/下的CMakeLists.txt。

4. 设置环境变量:

export MKL_DYNAMIC=FALSE
export MKL_NUM_THREADS=1

其中MKL_NUM_THREADS用来配置使用的线程数，线程数的设置一般取2-3比较好，具体需要根据机器的具体情况 。

5. 执行下面的命令，其中-DMKL_ROOT是指向MKL的路径：

mkdir build
cd build
cmake .. -DMKL_ROOT=/nfs/disk/perm/tools/intel/parallel_studio_xe_2013/composer_xe_2013_sp1.0.080/mkl/
make –j 2

6. 同普通编译4

2.4 EVALB

RNNG 中在做准确率的计算的时候会用到EVALB模块，从http://nlp.cs.nyu.edu/evalb/上下载EVALB.tgz，解压后，放在rnng目录下即可。

3. Prepare Data

经过以上 4 步，基本将代码准备好了，然而在运行之前我们还需要将数据准备好。如论文中所说，使用 Penn Treebank§2–21 作为训练集，§24 作为验证集，§23 作为测试集。本文档使用的数据集在/work4/zhangsy/rnng/wsj 目录下，其中有 00-24 目录，每个中有若干 mrg 文件，每个文件中有几个结构化的句法树，如下图所示。

```
[zhangsy@grid-0 wsj]$ ls
00  02  04  06  08  10  12  14  16  18  20
01  03  05  07  09  11  13  15  17  19  21
[zhangsy@grid-0 wsj]$ cd 00
[zhangsy@grid-0 00]$ ls
wsj_0001.mrg  wsj_0014.mrg  wsj_0027.mrg  ws
wsj_0002.mrg  wsj_0015.mrg  wsj_0028.mrg  ws
wsj_0003.mrg  wsj_0016.mrg  wsj_0029.mrg  ws
wsj_0004.mrg  wsj_0017.mrg  wsj_0030.mrg  ws
wsj_0005.mrg  wsj_0018.mrg  wsj_0031.mrg  ws
wsj_0006.mrg  wsj_0019.mrg  wsj_0032.mrg  ws
wsj_0007.mrg  wsj_0020.mrg  wsj_0033.mrg  ws
wsj_0008.mrg  wsj_0021.mrg  wsj_0034.mrg  ws
wsj_0009.mrg  wsj_0022.mrg  wsj_0035.mrg  ws
wsj_0010.mrg  wsj_0023.mrg  wsj_0036.mrg  ws
wsj_0011.mrg  wsj_0024.mrg  wsj_0037.mrg  ws
wsj_0012.mrg  wsj_0025.mrg  wsj_0038.mrg  ws
wsj_0013.mrg  wsj_0026.mrg  wsj_0039.mrg  ws
[zhangsy@grid-0 00]$ more wsj_0001.mrg

( (S
    (NP-SBJ
      (NP (NNP Pierre) (NNP Vinken) )
      (, ,)
      (ADJP
        (NP (CD 61) (NNS years) )
        (JJ old) )
      (, ,) )
    (VP (MD will)
      (VP (VB join)
        (NP (DT the) (NN board) )
```

图二：文档中使用数据概览

3.1 Preprocess

在 wsj 目录下的数据是分离的，而且格式是树状的，这一步将把数据整合起来，并按照每个句法树一行来存储。

1. 在 rnng 目录下新建 prepocess.py 文件，写入如下内容：

```
import os
import sys

def convert_to_one_line(file):
    lines = open(file, 'r').read().split('\n')
    sens = []
    sen = ""
    for line in lines:
        if line:
            if line[0] == '(' and sen:
                sen = sen[1:-2].strip() + '\n'
                sens.append(sen)
                sen = ""
            line = line.strip()
            if line:
```

```
            sen += '{} '.format(line)
    if sen:
        sen = sen[1:-2].strip() + '\n'
        sens.append(sen)

    return ''.join(sens)[:-1]

def convert(wsj):
    dirs_map = {
        "train": ['02', '03', '04', '05', '06', '07', '08', '09', '10', '11', '12', '13', '14', '15', '16', '17', '18',
                  '19', '20', '21'],
        "dev": ['24'],
        "test": ['23']
    }
    for dataset in ['train', 'dev', 'test']:
        dirs = dirs_map[dataset]
        with open("{}.all".format(dataset), 'a') as f:
            for dir in dirs:
                dir = "{}/{}".format(wsj, dir)
                files = sorted(os.listdir(dir))
                for file in files:
                    f.write(convert_to_one_line("{}/{}".format(dir, file)) + '\n')

def extract_unk_lines(file):
    """
    extract the lines contain 'UNK' in train.oracle to train.txt, which will be used in cluster
    """
    f = open(file, 'r')
    lines = f.read().split('\n\n')[:-1]
    f.close()
    for line in lines:
        items = line.split('\n')
        print items[4]

def extract_stemmed_trees(file):
    """
    extract the lines of stemmed trees in *.oracle to *.stem, which will be used in evaluation
    """
    lines = open(file).read().split('\n')
    for line in lines:
        if len(line) > 1 and line[0] == '#':
            print line[2:]

if __name__ == '__main__':
    if len(sys.argv) < 2:
        print "please input wsj dir!"
        exit()
    convert(sys.argv[1])
```

2. 执行命令 *python2 preprocess.py /work4/zhangsy/rnng/wsj*，在 rnng 目录下得到 train.all, dev.all, test.all 三个文件。

3. 在 rnng 下新建 data 文件夹，将三个文件移动到 data 目录下。

4. 一个需要处理的小细节：在 train.all 中有一行是

(NP (NP (QP (# #) (CD 200) (CD million)) (-NONE- *U*)) (PP (IN of) (NP (NP (JJ undated) (JJ variable-rate) (NNS notes)) (VP (VBN priced) (NP (-NONE- *)) (PP-CLR (IN at) (NP (JJ par))) (PP (IN via) (NP (NNP Merill) (NNP Lynch) (NNP International) (NNP Ltd)))))) (. .))

这行句子的第一个词是#，会影响之后的操作，因此把第一个词删除，改为:

(NP (NP (QP (CD 200) (CD million)) (-NONE- *U*)) (PP (IN of) (NP (NP (JJ undated) (JJ variable-rate) (NNS notes)) (VP (VBN priced) (NP (-NONE- *)) (PP-CLR (IN at) (NP (JJ par))) (PP (IN via) (NP (NNP Merill) (NNP Lynch) (NNP International) (NNP Ltd)))))) (. .))

3.2 Format

模型的输入数据需要有一定的格式，对于 discriminative model 的数据每个句子需要被表示成五部分：句法树，原始的句子，小写的句子，加入 unknown 词的句子，actions；对于 generative model 的数据每个句子需要被表示成四部分：句法树，原始句子，加入 unknown 词的句子，actions。步骤如下：

1. 修改 rnng 下的 get_oracle.py 和 get_oracle_gen.py 两个代码。因为我使用的数据中 nonterminal tokens 种类较多，需要做一个 stemming 的工作，将类似 “NP-SBJ” 这样的 nonterminal token 中 ’ - ‘ 后的部分去掉，变成 “NP” 。修改后的代码为 /work4/zhangsy/rnng 目录下的 get_oracle_stem.py 和 get_oracle_gen_stem.py。

特别说明：由于代码比较长，不在文档出贴出。一般如果读者使用 Penn Treebank 比较早期的版本将不存在这个问题，直接使用代码库中的 get_oracle.py 和 get_oracle_gen.py 即可。如有代码需要可联系邮件联系我。

2. 执行如下 6 个命令：

python get_oracle_stem.py data/train.all data/train.all > data/train.oracle
python get_oracle_stem.py data/train.all data/dev.all > data/dev.oracle
python get_oracle_stem.py data/train.all data/test.all > data/test.oracle

python get_oracle_gen_stem.py data/train.all data/train.all > data/train_gen.oracle
python get_oracle_gen_stem.py data/train.all data/dev.all > data/dev_gen.oracle
python get_oracle_gen_stem.py data/train.all data/test.all > data/test_gen.oracle

3. 将 dev.oracle 和 test.oracle 中的句法树（也就是每个句子的第一行去除#）单独输出到文件 dev.stem 和 test.stem，以备后续之用。代码参见 preprocess.py 中的 extract_stemmed_trees 函数。

3.3 Word cluster

在 generative model 中需要用到词的聚类，论文中采用的是 Brown Cluster。这里简单介绍如何生成聚类文件：

1. 将 train.oracle 中，带有’ UNK’ 的句子（也就是每个句子的第四种表示），单独输出到文件 train.txt。代码参见 preprocess.py 中的 extract_unk_lines 函数。

2. 下载 brown-cluster 代码库 *git clone https://github.com/percyliang/brown-cluster.git*, 执行 make 命令编译代码。

3. 在 brown-cluster 目录下，执行命令 *./wcluster --text train.txt --c 156* ，其中 156 是类别的个数，等于 sqrt(V)，V 是词的个数。

4. 输出的文件为 train-c156-p1.out/paths，将其重命名为 word_clusters.txt，放在 rnng/data 目录下备用。

4. Discriminative Model

Discriminative model 是预测句法树的模型，模型最终输出的是预测到的句法树。可以认为模型建模的是给定句子，句法树的条件概率 p(y|x)。rnng/nt-parser 目录下的 nt-parser.cc 为 discriminative 模型的代码。

4.1 Modify

在运行代码之前，为了适应我们服务器的配置，需要对 nt-parser.cc 代码作出一些修改：

1. 将"/tmp/parser_dev_eval." 改为 "tmp/parser_dev_eval."，并在 rnng 目录下新建一个 tmp 目录。
2. 将 python remove_dev_unk.py 改为 python2 remove_dev_unk.py。
3. 重新编译代码，执行命令：

cd build
make –j 2

4.2 Configuration

模型的输入参数有如下：

```
Configuration options:
  -T [ --training_data ] arg          List of Transitions - Training corpus
  -x [ --explicit_terminal_reduce ]   [recommended] If set, the parser must
                                      explicitly process a REDUCE operation to
                                      complete a preterminal constituent
  -d [ --dev_data ] arg               Development corpus
  -C [ --bracketing_dev_data ] arg    Development bracketed corpus
  -p [ --test_data ] arg              Test corpus
  -D [ --dropout ] arg                Dropout rate
  -s [ --samples ] arg                Sample N trees for each test sentence
                                      instead of greedy max decoding
  -a [ --alpha ] arg                  Flatten (0 < alpha < 1) or sharpen (1 <
                                      alpha) sampling distribution
  -m [ --model ] arg                  Load saved model from this file
  -P [ --use_pos_tags ]               make POS tags visible to parser
  --layers arg (=2)                   number of LSTM layers
  --action_dim arg (=16)              action embedding size
  --pos_dim arg (=12)                 POS dimension
  --input_dim arg (=32)               input embedding size
  --hidden_dim arg (=64)              hidden dimension
  --pretrained_dim arg (=50)          pretrained input dimension
  --lstm_input_dim arg (=60)          LSTM input dimension
  -t [ --train ]                      Should training be run?
  -w [ --words ] arg                  Pretrained word embeddings
  -b [ --beam_size ] arg (=1)         beam size
  -h [ --help ]                       Help
```

4.3 Train

训练 discriminative model，执行下面的命令：

./build/nt-parser/nt-parser -x -T data/train.oracle -d data/dev.oracle -C data/dev.stem -P -t --input_dim 128 --lstm_input_dim 128 --hidden_dim 128 -D 0.2

模型训练过程不会主动停止，需要人工停止。训练过程中每次 update100 个句子，约耗时 100ms 左右，每隔 15 次 update 会在验证集上评估一次效果，输出预测的 F1 值。如果 F1 值大于最好的 F1 值，则将这次的模型存入 ntparse_XXX-pidXXX.params 文件中。大约需要跑 11 轮以上可以达到比原论文更好的效果。

4.4 Test

测试 discriminative model，执行下面的命令，参见/work4/zhangsy/rnng/test.sh：

.build/nt-parser/nt-parser -x -T data/train.oracle -d data/dev.oracle -C data/test.stem -m latest_model -P -p data/test.oracle --input_dim 128 --lstm_input_dim 128 --hidden_dim 128 -D 0.2

注意-m 参数可以设置最佳模型的*.params 文件，也可以设置为指向最佳模型的软链接 lastest_model. 测试最终会输出在测试集上的 F1 值，最佳可达到 92.32 左右。

5. Generative Model

Generative model 是生成模型，在预测 action 的同时要生成词。可以认为模型建模的是句子和句法树之间的联合概率 p(x,y)。

5.1 Modify

在运行代码之前， 需要对原始的代码作出一些修改：

1. 将latest_model改为latest_model_gen
2. 重新编译代码，执行命令：

cd build
make –j 2

5.2 Configuration

模型的输入参数有如下：

```
Configuration options:
  -T [ --training_data ] arg          List of Transitions - Training corpus
  -x [ --explicit_terminal_reduce ]   [not recommended] If set, the parser must
                                      explicitly process a REDUCE operation to
                                      complete a preterminal constituent
  -D [ --dropout ] arg                Use dropout
  -c [ --clusters ] arg               Clusters word clusters file
  -d [ --dev_data ] arg               Development corpus
  -p [ --test_data ] arg              Test corpus
  -e [ --eta_decay ] arg              Start decaying eta after this many epochs
  -m [ --model ] arg                  Load saved model from this file
  --layers arg (=2)                   number of LSTM layers
  --action_dim arg (=16)              action embedding size
  --input_dim arg (=32)               input embedding size
  --hidden_dim arg (=64)              hidden dimension
  --pretrained_dim arg (=50)          pretrained input dimension
  --lstm_input_dim arg (=60)          LSTM input dimension
  -t [ --train ]                      Should training be run?
  -w [ --words ] arg                  Pretrained word embeddings
  -h [ --help ]                       Help
```

5.3 Train

训练 generative model，执行下面的命令：

./build/nt-parser/nt-parser-gen -x -T data/train_gen.oracle -d data/dev_gen.oracle -c data/word_clusters.txt -t --input_dim 256 --lstm_input_dim 256 --hidden_dim 256 -D 0.3

模型训练过程不会主动停止，需要人工停止。训练过程中每次 update100 个句子，约耗时 300ms 左右，每隔 100 次 update 会在验证集上评估一次效果，输出在验证集上的 ppl。如果 ppl 值小于最好的 ppl 值，则将这次的模型存入 ntparse_gen_XXX-pidXXX.params 文件中。大约需要跑 16 轮以上可以达到比原论文更好的效果。

5.4 Test

由于生成模型建模的是联合概率 p(x, y)，为了评估其准确率和作为 language model 的效果，都需要求出边缘概率 p(x)。因此采取先从 discriminative model 中采样，在利用 generative model 重新排序的方法。具体请见原论文。步骤如下：

1. 从 discriminative model 中采样，对每个在测试集中的句子采样 100 个预测出的句法树，执行下面的命令：

./build/nt-parser/nt-parser -x -T data/train.oracle -d data/dev.oracle -C data/test.stem -m latest_model -P -p data/test.oracle --input_dim 128 --lstm_input_dim 128 --hidden_dim 128 -D 0.2 -s 100 -a 0.8 > test-samples.props

2. 去除 test-samples.props 中每一行的多余部分，执行命令 *utils/cut-corpus.pl 3 test-samples.props > test-samples.trees*

3. 从生成模型中获得联合概率，执行命令：

./build/nt-parser/nt-parser-gen -x -T data/train_gen.oracle --clusters data/word_clusters.txt --input_dim 256 --lstm_input_dim 256 --hidden_dim 256 -p test-samples.trees -m latest_model_gen > test-samples.likelihoods

4. 获得边缘概率，执行命令：

utils/is-estimate-marginal-llh.pl 2416 100 test-samples.props test-samples.likelihoods > llh.txt 2> rescored.trees

5. 执行以下四个命令：

utils/add-fake-preterms-for-eval.pl rescored.trees > rescored.preterm.trees
utils/replace-unks-in-trees.pl data/test.oralce rescored.preterm.trees > hyp.trees
python2 utils/remove_dev_unk.py data/test.stem hyp.trees > hyp_final.trees
EVALB/evalb -p EVALB/COLLINS.prm data/test.stem hyp_final.trees > parsing_result.txt

6. 以上命令可以集合成一个脚本，参见/work4/zhangsy/rnng/test-gen.sh。

llx.txt 文件中的最后几行会给出 language model 中边缘概率 p(x)的 perplexity， 大约可以达到 88.66。parsing_result.txt 中给出了 generative model 的准确率，F1 值大约为 92.88。

6. Reference

[1] Dyer C, Kuncoro A, Ballesteros M, et al. Recurrent Neural Network Grammars[J]. 2016.
[2] https://github.com/clab/rnng
[3] http://dynet.readthedocs.io/en/latest/install.html