RNNG Document

2016.10.31 张诗悦

1. Introduction

这篇文档将简单介绍 RNNG 源代码，以及其依赖库 CNN 的代码。

RNNG 的具体内容请参见 Chris Dyer 等在 NAACL 2016 上发表的论文：Recurrent Neural Network Grammars[1]，代码库为：https://github.com/clab/rnng，代码的使用可以参见笔者之前写的 Rnng Code Use Giude[4] 。推荐读者在成功运行代码，了解代码运用流程后，再来读本文档。本文档中将介绍笔者对 RNNG 代码的理解，以辅助读者对代码的阅读。CNN 是 RNNG 的一个依赖库，也是 Chris Dyer 实验室编写的深度学习通用模块的代码库。CNN 的代码可以在编译 RNNG 代码同时得到编译，也可以单独编译运行（https://github.com/clab/cnn-v1）。现在已经封装成了 toolkit: DyNet (https://github.com/clab/dynet)，笔者写这篇文档的时候，DyNet 刚刚获得 license，因此文档还非常不完整。本文档将介绍 CNN 库中核心模块的运用，以及笔者对核心模块代码的理解，以辅助读者使用 CNN 库和 理解 RNNG 代码。

以下内容均为笔者自己的理解，错误的地方还望各位读者批评指正。

2. CNN

2.1. Cores

这一部分将介绍 CNN 中一些常用的模块代码，并不完整，后续将逐渐修改和补全。

❖ **基本数据结构模块**

1. dim.h, dim.cc
   - struct Dim: 表示数据 shape 的结构体
     - 变量
       - int d[CNN_MAX_TENSOR_DIM]: 存储每一维的 size
       - int nd: 代表总的维度，最大是 7
       - int bd: 代表 batch 的数量
     - 函数
       - unsigned int size():返回总的 size
       - unsigned int batch_size(): 返回一个 batch 的 size
       - unsigned int sum_dims(): 总的 dim，也就是把每一位的长度相加
       - Dim truncate()：简化，去除长度为 1 的维度
       - Dim single_batch() : 设置为一个 batch
       - void resize(unsigned int i): 重新设置维度个数
       - unsigned int ndims()： 返回维度数，即返回 nd
       - unsigned int rows(): 行数，返回第一维的长度
       - unsigned int cols(): 列数，nd>1 时，返回第二维的长度，如果 nd==1, 返回 1
       - unsigned int batch_elems(): batch 数，返回 bd

        - void set(unsigned int i, unsigned int s):设置第 n 维的维度
        - unsigned int size(unsigned int i): 返回第 n 维的维度
        - Dim transpose(): 转置，只有在 nd<=2 的时候合法
        - 另外还有一些运算符的重载：[], ==, !=, <<, >>
2. tensor.h tensor.cc
    - struct Tensor: 定义 tensor 的结构体
        - 变量
            - Dim d: Dim 类型的对象，定义了 tensor 的 shape
            - float* v: float 类型的指针，tensor 中的值是按照 vector 的形式存储的，v 指向其第一个节点
            - std::vector<Tensor> bs: 一组 tensor，每个对应着一个 batch
        - 函数
            - Tensor batch_elem(unsigned b): 获取第 b 个 batch 的 tensor
            - std::vector<Tensor> batch_elems(): 获取所有的 batches
            - 还有一系列函数是利用 Eigen 定义的，主要功能是完成从 tensor 向 matrix 的转化，向 vector 的转化，以及将每个 batch 内行（或列）首尾相接得到的 matrix
    - struct TensorTools: 对 tensor 做初始化等操作的结构体， 包括：按照均匀、高斯、伯努利分布进行随机初始化；初始化为 0，或者某个常数；访问、设置、复制 tensor 中的某个元素。
    - real as_scalar(const Tensor &t): 把一个 tensor 转化成一个数（tensor 的 size 必须是 1）
    - std::vector<real> as_vector(const Tensor &v): 把一个 tensor 转化为一个 vector，也就是把 tensor 的变量 v 返回
    - real rand01(): 按照均匀分布在 0 到 1 之间随机出一个数
    - int rand0n(int n)：在 0，n 之间随机出一个整数
    - real rand_normal(): 按照高斯分布随机出一个 0 到 1 之间的数
3. dict.h dich.cc
    - class Dict: 字典类型
        - 变量
            - bool frozen: 设置是否把字典锁定不可更改
            - bool map_unk: 是否把 unknown 的词设置为 unk_id
            - int unk_id: unknown 的词的 id
            - std::vector<std::string> words_: 所有的词
            - Map d_: word-id map
        - 函数
            - unsigned size(): 返回 words_的长度
            - bool Contains: 是否存在某个词
            - void Freeze(): 设置字典不可更改
            - bool is_frozen()：返回 frozen
            - int Convert(const std::string& word): 把词的转化为 id，如果没有这个词，就添加到词表 words_里
            - std::string& Convert(const int& id): 把 id 转化为词

        - void SetUnk(const std::string& word): 在字典锁定之后，设置 map_unk=true
    - ReadSentence: 把一个句子转化为词 id 的序列
    - ReadSentencePair：把一对句子转化为词 id 的序列，两句子用 “|||” 分割

❖ **模型构建模块**

1. cnn.h cnn.cc
    - struct ComputationGraph: 计算网络结构体
        - 变量
            - std::vector<Node*> nodes：存储计算网络中所有的节点，按照添加节点的时间顺序存储
            - std::vector<VariableIndex> parameter_nodes：nodes 的一个子集，存储计算网络中需要更新参数的节点
            - ExecutionEngine* ee：执行引擎，用于整个计算网络的计算操作，例如:前向传播和反向传播
        - 函数
            - add_input: 增加输入变量
            - add_parameters: 增加模型参数
            - add_lookup: 增加 lookup 类型的模型参数
            - add_function: 增加函数
            - forward: 前向传播计算，从第一个节点到最后一个
            - incremental_forward: 增量的前向传播计算，从上一个计算过的节点到最后一个
            - get_value(i): 获得某个变量的值
            - backward: 计算反向传播的梯度
            - backward(i): 计算某个节点反向传播的梯度
    - struct Node：计算节点的基结构体。Node 有很多子类，例如 Min, Max, DotProduct 等，具体参见 nodes.h, param-nodes.h。子类都需要实现自己的 forward_impl 和 backward_impl 函数。
        - 函数
            - forward: 前向传播的计算，其中调用 forward_impl
            - backward：后向传播的计算, 其中调用 backward_impl
            - forward_impl: 具体执行前向传播的计算，虚函数，由子类实现
            - backward_impl: 具体执行后向传播的计算，虚函数，由子类实现
2. expr.h expr.cc
    - struct Expression: 表达式类，cnn 框架下，模型运算过程中所有的计算单元都是表达式（expression），如参数，输入，中间结果。

    例如：

```
ComputationGraph cg;
Expression W = parameter(cg, m.add_parameters({HIDDEN_SIZE, 2}));  //参数
vector<cnn::real> x_values(2);
Expression x = input(cg, {2}, &x_values); //输入
Expression h = tanh(W*x);  //模型中的运算
```

        - 变量

        - ComputationGraph *pg: 表达式一定是来自于某一个计算网络
        - VariableIndex i：该表达式对应着计算网络中的某个变量，这里就是变量的 index
    - 函数
        - Expression(ComputationGraph *pg, VariableIndex i)：表达式在构建的时候，需要给定计算网络，和要表达的变量的索引
        - const Tensor& value() const:获取表达式的值，返回表达式表示变量的值
    - 声明表达式函数，以及进行表达式之间的运算函数。简单举几个例子，其他类似。
        - input: 定义输入变量
        - const_parameter: 定义常数参数,参数不更新
        - parameter：定义参数
        - lookup: 定义 LookupParameters 类型的参数
        - const_look: 定义常数 LookupParameters 类型的参数，参数不更新
        - zeros：定义归 0 的函数
        - nobackprop: 定义不需要求导的函数
        - +，-，*：定义一些系列表达式之间的操作符
        - ……
3. model.h model.cc
    - struct ParameterBase: 虚基结构体
        - 函数
            - squared_l2norm: 参数的 l2 正则
            - g_squared_l2norm：参数导数的 l2 正则
            - size()：参数的大小，也就是参数的个数
    - struct Parameters：普通参数的结构体（例如，权重矩阵等）
        - 变量
            - dim: 参数的维度
            - values：参数的值，tensor
            - g: 参数的导数，tensor
        - 函数
            - 构造函数：初始化 values 和 g，values 是随机初始化，g 全部初始化为 0
            - scale_parameters: 所有的参数上同乘以一个常数
            - size: dim 的 size
            - accumulate_grad：累加导数
    - struct LookupParameters：用于 embed 离散对象参数的结构体，参数是一组 tensor，每个对应着一个需要 embedding 的对象（例如，词的 embedding 矩阵）
        - 变量
            - dim: 参数的维度，指每个 tensor 的维度
            - values：参数的值，vector<tensor>
            - grads: 参数的导数，vector<tensor>

        - non_zero_grads: 记录不为 0 的导数
    - 函数
        - 构造函数：初始化 values 和 g，values 是随机初始化，g 全部初始化为 0
        - scale_parameters: 所有的参数上同乘以一个常数
        - size: values 的 size×dim 的 size
        - accumulate_grad：每个 tensor 内部进行导数累加
- class Model：模型通用类，任何模型都是一个 Model 类型的对象
    - 变量
        - all_params: ParameterBase 类的参数集合，也就是 params 和 lookup_params 的集合，模型中所有的参数
        - params: Parameters 类的参数集合
        - lookup_params: LookupParameters 类的参数集合
        - gradient_norm_scratch: 模型内参数的 l2 正则
    - 函数
        - gradient_l2_norm：计算模型中导数的 l2 正则
        - add_parameters: 添加 Parameters 类的参数
        - add_lookup_parameters: 添加 LookupParameters 类的参数
        - save, load: 保存和加载模型

4. training.h training.cc
    - struct Trainer：训练工具的基结构体
        - 变量
            - eta0, eta…: 学习率参数
            - lambda: 正则项参数
            - clipping_enabled…: 对梯度进行 scale 的参数
            - model: 要训练的模型
        - 函数
            - update: 更新模型参数的函数
            - update_epoch: 更新学习率的函数
            - clip_gradients: scale 梯度的函数
    - struct SimpleSGDTrainer
    - struct MomentumSGDTrainer
    - struct AdagradTrainer
    - struct AdadeltaTrainer
    - struct RmsPropTrainer
    - struct AdamTrainer

以上的优化方法都需要分别实现自己的update方法。在每个update方法中，要分别更新模型的两部分参数：Parameters和LookupParameters. 对于每个Parameters整体更新即可，而对于每个LookupParameters则需要逐列更新。

❖ 深度学习模型

1. rnn.h rnn.cc

- struct RNNbuilder: 实现 RNN, LSTM，GRU 的接口结构体

- 变量
  - cur: 当前状态的 timestamp
  - head: 记录历史的timstamp
  - sm: rnnbuilder当前的工作状态：new_graph, start_new_sequence, add_input
- 函数
  - state: 返回cur
  - new_graph: 在计算网络cg中添加新的rnnbuilder
  - start_new_sequence: 开始新的句子，并用h_0初始化timestamp 0，这个函数需要在add_input之前和new_graph之后调用
  - add_input: 增加新的timestamp输入，返回输出的隐向量，recurrent的链接位置为cur, 即新添加的节点要recurrent到当前节点。
  - add_input(prev, x): 增加新的timestamp输入，返回输出的隐向量，但是recurrent的链接位置为prev, 而非cur。
  - rewind_one_step: 后退一步，将cur设置为上一步的状态
- 虚函数：需要子类去实现
  - back: 返回当前状态cur对应的隐向量
  - final_h: 返回rnn最后输出的隐向量
  - get_h: 返回第i个timestamp输出的隐向量
  - final_s: 返回rnn最后输出的状态
  - get_s: 返回第i个timestamp的状态
  - copy: 拷贝函数
  - num_h0_components
  - new_graph_impl
  - start_new_sequence_impl
  - add_input_impl

- struct SimepleRNNBuilder: 继承 rnnbuilder, 标准的 rnn 模型
  - 变量
    - params: 模型参数，Parameters类型，第一个index是层数
    - param_vars: 模型参数，Expression类型，也就是params中的参数加入到计算网络后的表达式对象，第一个index是层数
    - h: 记录每个timestamp每层的隐向量， 第一个index是timestamp
    - h0：初始化的隐状态
    - layers: 模型的层数
    - lagging: 是否使用滞后
  - 函数
    - SimpleRNNBuilder: 构造函数，为model模型添加rnn相关的参数，初始化params
    - new_graph_impl: 把参数添加到计算网络cg中，返回Expression类的对象，初始化param_vars
    - start_new_sequence_impl: 用参数h_0初始化h0
    - add_input_impl: 执行向前一个timestamp的操作，循环每一层执行:
      1. x=h(cur) =tanh(W*x + U*h(prev) +b)

2. 返回当前timestamp的隐向量
- add_auxiliary_input: 执行向前一个timestamp的操作，但除了输入in之外，还多出一个aux作为输入，循环每一层执行:
  1. x=h(cur)=tanh(W*x + U*h(prev) + L*aux + b)
  2. 返回当前timestamp的隐向量

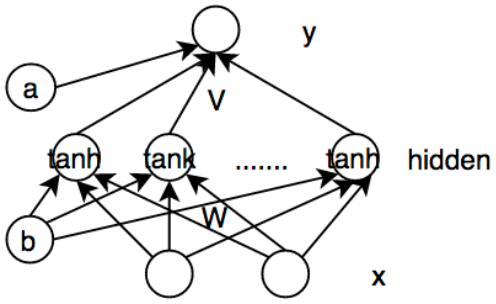
2. lstm.h lstm.cc

- struct LSTMBuilder: 继承 RNNBuilder，lstm 模型
  - 变量
    - params: 模型参数，Parameters类型， 第一个index是层数
    - param_vars: 模型参数，Expression类型，也就是params中的参数加入到计算网络后的表达式对象，第一个index是层数
    - h, c：记录每个timestamp每层的隐向量和状态， 第一个index是timestamp
    - has_initial_state: 是否有初始化的状态，如果等于false，则h0和c0为0
    - layers: 模型层数
    - dropout_rate: dropout的比例
  - 函数
    - LSTMBuilder: 构造函数，为model模型添加rnn相关的参数，初始化params
    - new_graph_impl: 把参数添加到计算网络cg中，返回Expression类的对象，初始化param_vars
    - start_new_sequence_impl: 用参数hinit初始化h0，c0
    - add_input_impl: 执行向前一个timestamp的操作，循环每一层执行:
      1. 如果有初始化，取出该层初始化的h0和c0
      2. 如果有dropout, 对输入dropout
      3. i = logistic(bi + Wi*in + Ui*h(prev) + Ci*c(prev))
      4. f = 1 - i
      5. w = tanh(bc + Wc*in + Uc*h(prev)
      6. c(cur) = i×w + f×c(prev)
      7. o = logistic(bo + Wo*in + Uo*h(prev) + Ci*c(cur))
      8. h(cur) = o×tanh(c(cur))
      9. 如果有dropout, 对输出的h(cur)dropout
      10. 返回h(cur)

2.2. Examples

上一部分简单地介绍了 cnn 中的一些核心模块，有些地方比较复杂，也不直观。那么这一部分，将介绍一些简单的例子，增强对核心模块的理解和运用。这些例子在 cnn 代码库的 examples 目录下，读者可以在那里看到具体的代码，并可以尝试运行看到输出的结果。

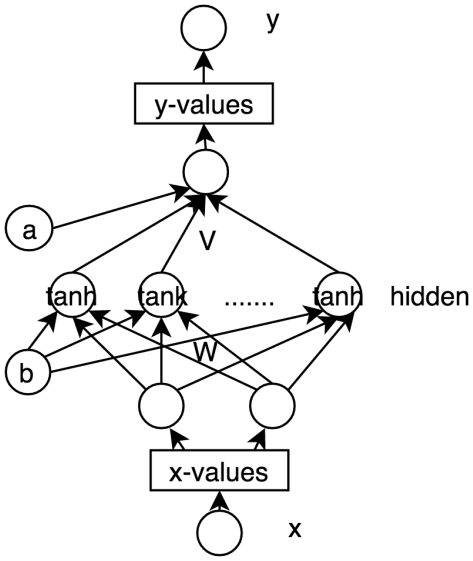
1. xor.cc：一个简单的两层的回归模型，模型的结构如图：



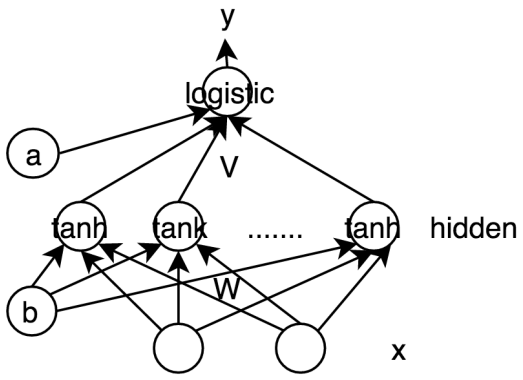
代码简介：

- main:
  - 定义隐层维度，迭代次数
  - 定义model, SimpleSGD作为模型优化算法：Model, SimpleSGD
  - 定义计算网络（ComputationGraph, cg）: ComputationGraph
  - 添加模型参数W, V, a, b, 以及输入x, y: parameter()
  - 构建模型，并把平方距离作为损失函数: input()
  - 训练模型，打印每轮的loss: forward(), backward()

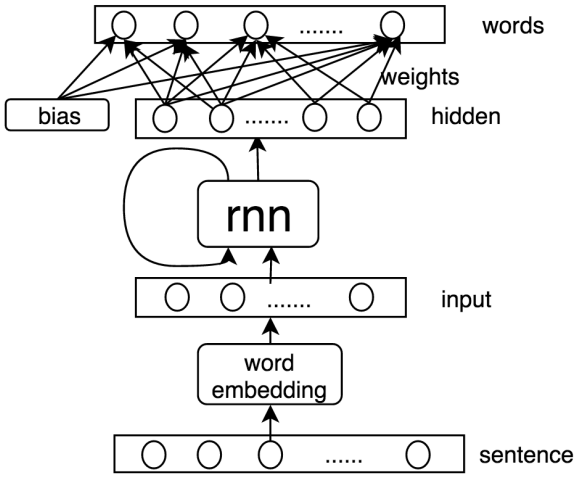
2. xor-batch.cc：模型结构和xor.cc相同，唯一不同的是在训练的时候，xor-batch.cc直接将所有样本输入，用Dim来设置batch，而不是像xor.cc中生一个样本更新一次。该代码中也设置一个样本作为一个batch, 但是训练时收敛的速度显然没有xor.cc快，不知为何，之后会再深入研究。
3. xor-batch-lookup.cc: 基本模型结构和xor.cc相同，但是分别在输入和输出层增加了embedding的矩阵，也就是两个LookupParameters。模型结构如下:



4. xor-xent.cc：基本模型结构也和xor.cc相同，不同之处是该模型为二分类模型，在最后一层增加了logistic激活单元，并把交叉熵作为损失函数，训练采用每个样本跟新一次的训练方式。模型结构如下：



5. rnnlm.cc, rnnlm2.cc: rnn语言模型，rnnlm2和rnnlm不同之处仅仅是打印了每步更新后真实词对应的模型输出值。 基本的模型结构如下：

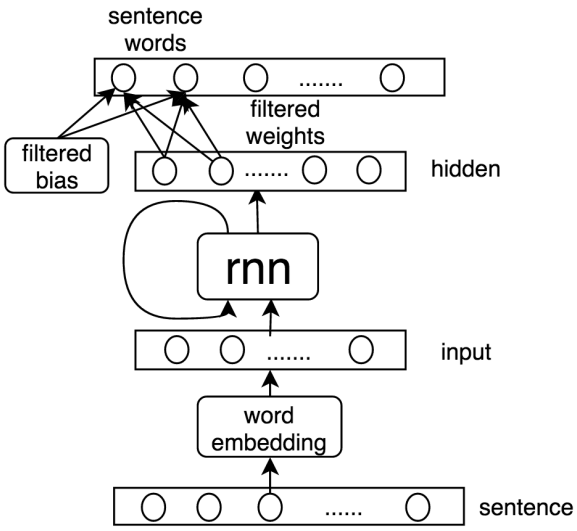


代码简介：

- RNNLanguageModel:
  - 变量
    - p_c: 对输入的词进行embedding的lookup参数
    - p_R: 隐向量向输出转化的权重矩阵
    - p_bias：隐向量向输出转化的偏置向量
  - 函数
    - BuildLMGraph: 建立计算网络，返回输出损失函数（logsoftmax之后正确词的概率）
    - RandomSample: 根据训练好的模型随机生成句子
- main:
  1) 定义句子的开始结束标签，定义训练集和测试集的矩阵
  2) 读取训练集和验证集的数据
  3) 定义模型存储文件名，定义model，simplesgd作为优化工具
  4) 用lstm作为rnn模型， 建立rnn语言模型

5) 训练，每次输入一个句子更新模型，每50个句子之后打印出当前训练集上的loss和ppl
6) 每500个句子在验证集上测试，并打印loss和ppl

6. rnnlm-batch.cc: 基本模型结构和rnnlm.cc一致，不同之处在于这里是以4个句子为一个batch来更新的，而不是逐个更新。
7. rnnlm-aevb.cc：该例子是Auto-Encoding Variational Bayes的实现代码。
8. rnnlm-givenbag: 基本模型结构和rnnlm.cc一致, 不同之处在于，该模型训练过程中，句子中有哪些词会“告知”模型，因此模型只需要从这些词中选择生成下个词即可，如图所示：



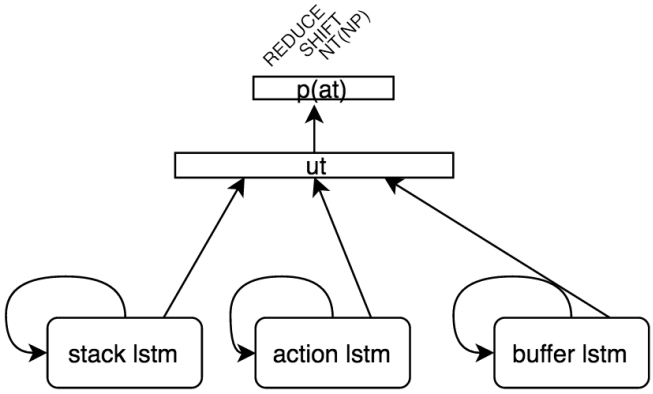
代码简介：

- RNNLanguageModel:
  - 函数
    - BuildLMGraph: 相比标准的rnnlm多了一个参数prows, 该参数记录了当前训练句子中所有的词。根据prows，过滤hidden->word的权重矩阵和偏移，只保留句子中有的词。也就是生成的词被控制在句子中存在的词范围内。
- main: 相比标准的rnnlm，多了rows，w2sl，rmsent三个向量，rows存储一个句子中不重复的词表，为w2sl用来辅助生成rows和rmsent，rmsent中存储一个句子中每个词在rows词表中对应的位置。

3. RNNG

3.1. Discriminative model

RNNG的discriminative model模型代码为nt-parser.cc。其基本模型结构如下图所示：



模型执行过程如下图：

Input: *The hungry cat meows .*

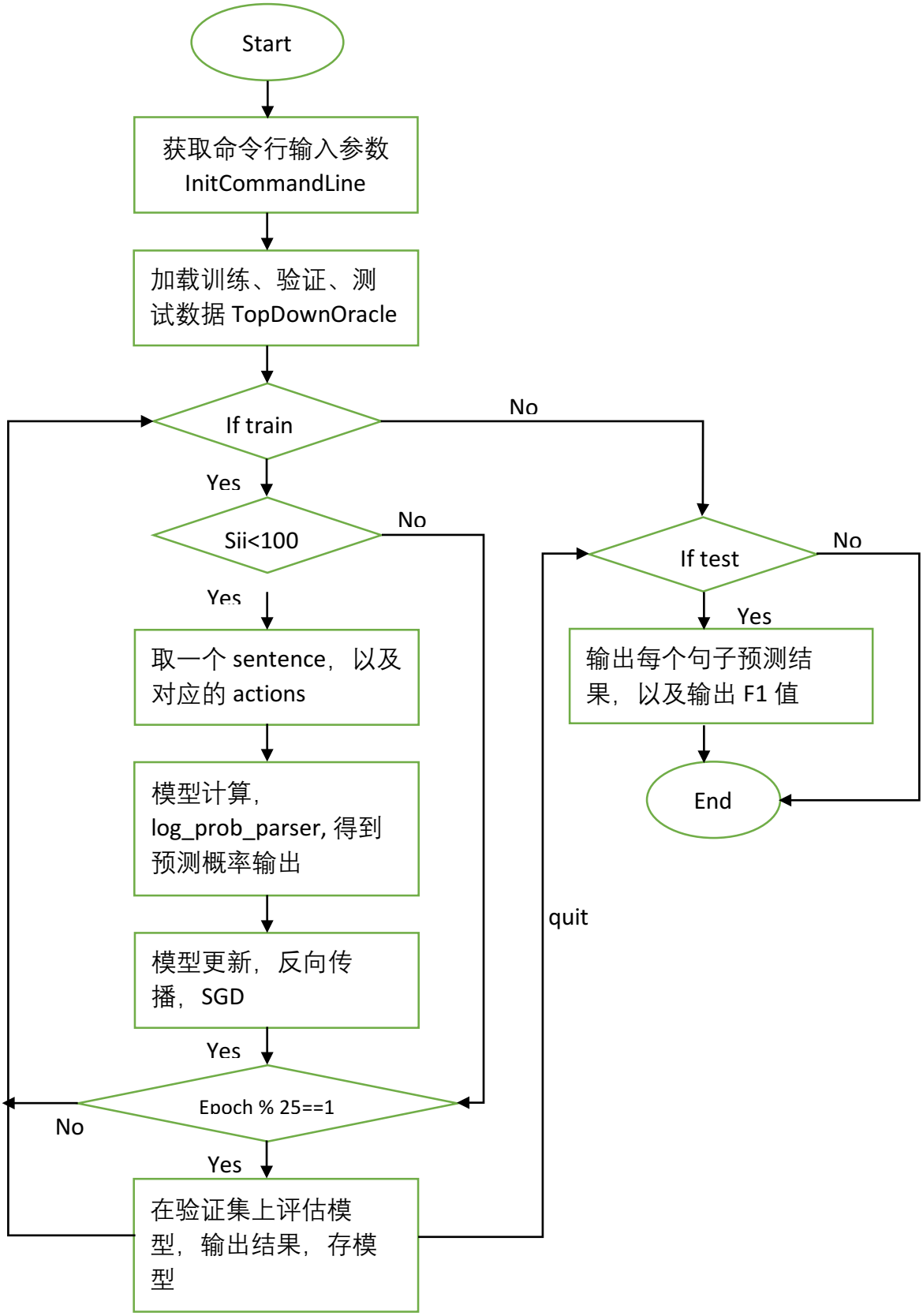
	Stack	Buffer	Action
0		*The* | *hungry* | *cat* | *meows* | .	NT(S)
1	(S	*The* | *hungry* | *cat* | *meows* | .	NT(NP)
2	(S | (NP	*The* | *hungry* | *cat* | *meows* | .	SHIFT
3	(S | (NP | *The*	*hungry* | *cat* | *meows* | .	SHIFT
4	(S | (NP | *The* | *hungry*	*cat* | *meows* | .	SHIFT
5	(S | (NP | *The* | *hungry* | *cat*	*meows* | .	REDUCE
6	(S | (NP *The hungry cat*)	*meows* | .	NT(VP)
7	(S | (NP *The hungry cat*) | (VP	*meows* | .	SHIFT
8	(S | (NP *The hungry cat*) | (VP *meows*	.	REDUCE
9	(S | (NP *The hungry cat*) | (VP *meows*)	.	SHIFT
10	(S | (NP *The hungry cat*) | (VP *meows*) | .		REDUCE
11	(S (NP *The hungry cat*) (VP *meows*) .)		

代码有两个主要部分：1）main函数： 对应着整体的训练和测试的流程；2）log_prob_parser函数：对应着模型的计算流程。下面将主要介绍这两个部分。

- Main函数

下图为 main 函数的主体执行流程，具体说明如下：

1. 调用 InitCommandLine 函数，从命令行获取输入参数；
2. 设置模型存储文件名，定义 model；
3. 加载训练集的数据和格式化前验证集句法树的文件路径，TopDownOracleh 类在 oracle.h 代码中定义，主要用来加载格式化的数据；
4. 如果有预先训练的 word embedding 参数传入，则读入 word embedding;
5. 把 term, action, nonterminal term, pos 的词表字典设置为不可改，如果验证集和测试集中有未知的词，要设置词典的 unk;

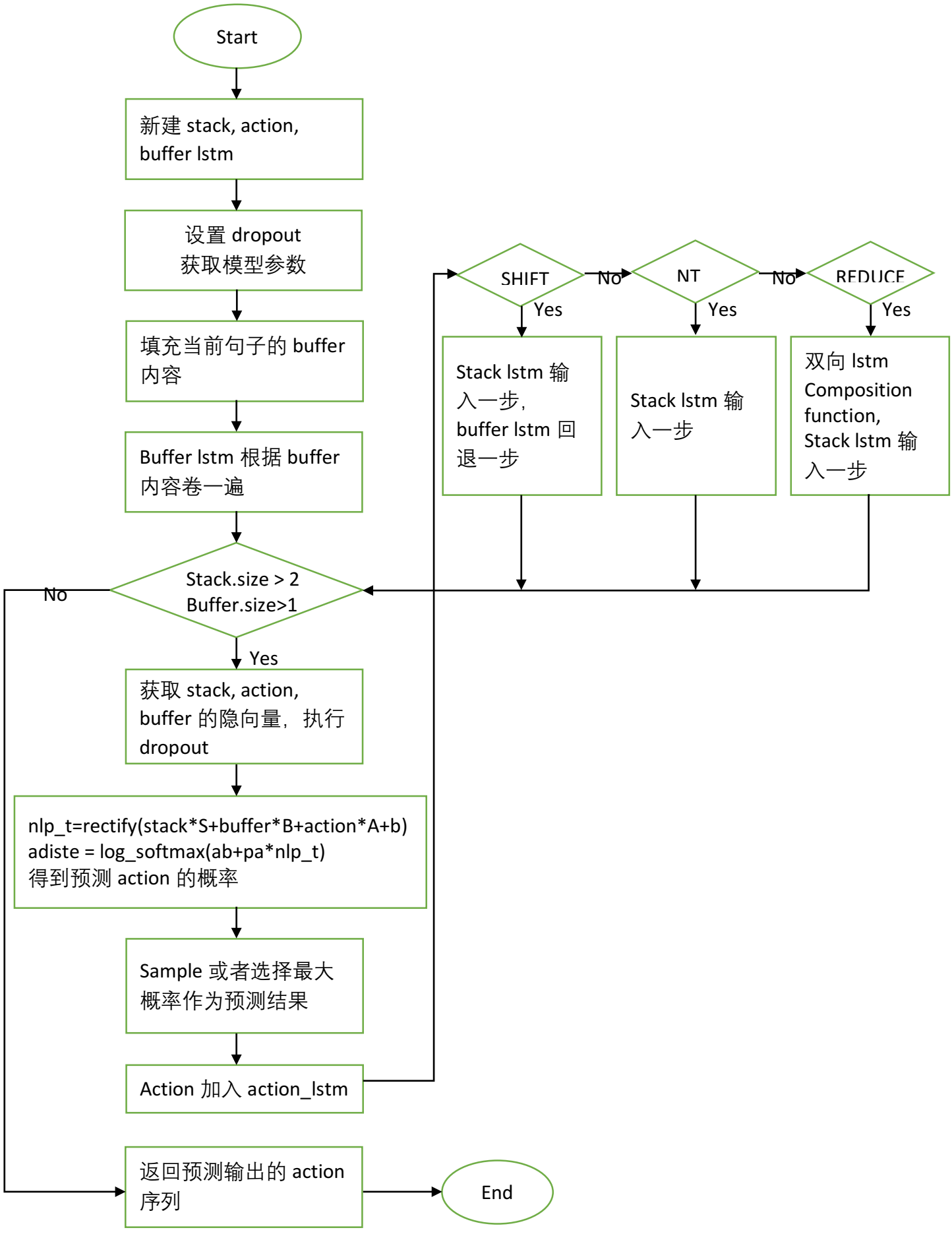

Start
获取命令行输入参数 InitCommandLine
加载训练、验证、测试数据 TopDownOracle
If train
No
Yes
Sii<100
No
Yes
取一个 sentence，以及对应的 actions
模型计算，log_prob_parser, 得到预测概率输出
模型更新，反向传播，SGD
Yes
Epoch % 25==1
No
Yes
在验证集上评估模型，输出结果，存模型
If test
No
Yes
输出每个句子预测结果，以及输出 F1 值
End
quit

6. 找出只出现一次的词，记录在 singleton 里；
7. 加载验证集和测试集的数据；
8. 设置 action 到 nonterminal word index 的对应数组；
9. 如果有训练好的 model 输入，则加载 model;
10. 如果需要训练，则进行 11 步，否则跳到 16 步；
11. 在没有接收到外部停止信号时，进行 12 步，否则跳到 16 步；
12. for 循环中依次更新 100 个句子，第一次进入循环时需要把全部训练集数据进行 shuffle, 每次取一个句子，取出对一个的 actions，传入 log_prob_parser 进行模型计算，反向传播更新模型；
13. 输出在训练集上的 ppl；
14. 每更新 25 轮，在验证集上评估一次，输出在验证集上的预测结果，和对应的 F1 值，如果新的 F1 值更优，则存储新的模型；
15. 返回 11 步；
16. 如果测试集不为空，进行 17 步，否则跳到 19 步；
17. 如果 N_SAMPLES 不为 0，说明需要用训练好的模型进行采样，输出采样结果；
18. 训练好的模型输出在测试集上的预测结果，和对应的 F1 值；
19. 结束。

❖ log_prob_parser函数

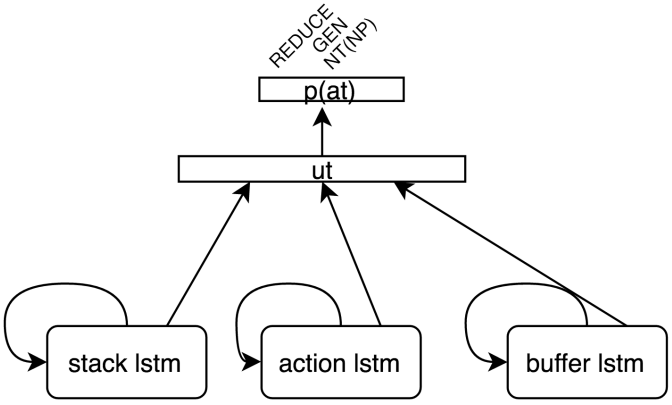
下图为 log_prob_parser 函数的主体执行流程，具体说明如下：

1. 新建 stack, buffer, action, fwd, rev 五个 lstm;
2. 设置是否使用 dropout;
3. 获取模型参数，action_lstm 输入初始的第一个 action；
4. 设置当前句子的 buffer 内容，而后 buffer lstm 执行一遍；
5. 如果 stack.size()>2 或者 buffer.size()>1, 则进行第 6 步，否则跳到 18 步
6. 挑选当前出合法的 actions;
7. 取出 stack lstm, action lstm, buffer lstm 当前的隐向量，nlp_t=rectify(stack*S+buffer*B+action*A+b)，adiste = log_softmax(ab+pa*nlp_t) 得到预测 action 的概率分布;
8. 如果需要 sample, 则 sample 出 action, 否则取概率最大的作为预测 action;
9. 如果在训练过程，记录当前预测是否正确，并按照正确的 action 继续；
10. action lstm 向前走一步；
11. 如果 action==SHIFT，进行 12 步，否则跳到 14 步；
12. stack lstm 向前走一步，buffer lstm 回退一步；
13. 如果 action==NT, 进行 14 步，否则跳到 15 步；
14. stack lstm 向前走一步；
15. 如果 action==REDUCE，进行 16 步，否则跳到 5 步；
16. 将 stack lstm 回退一个 children（也就是两个'()'之间的部分），用 fwd 和 rev 两个双向 lstm，得到 children 的双向隐向量，连接到一起，再做维度变换，得到 composed 输出再输入到 stack lstm 里；
17. 返回 5 步；
18. 结束，并返回预测的 action 序列。


Start
新建 stack, action, buffer lstm
设置 dropout
获取模型参数
填充当前句子的 buffer 内容
Buffer lstm 根据 buffer 内容卷一遍
Stack.size > 2
Buffer.size>1
No
Yes
获取 stack, action, buffer 的隐向量，执行 dropout
nlp_t=rectify(stack*S+buffer*B+action*A+b)
adiste = log_softmax(ab+pa*nlp_t)
得到预测 action 的概率
Sample 或者选择最大概率作为预测结果
Action 加入 action_lstm
返回预测输出的 action 序列
End
SHIFT
No
NT
No
REDUCE
Yes
Yes
Yes
Stack lstm 输入一步，buffer lstm 回退一步
Stack lstm 输入一步
双向 lstm Composition function, Stack lstm 输入一步

3.2. Generative model

RNNG的generative model模型代码为nt-parser-gen.cc。其基本模型结构如下图所示：



模型执行过程如下图：

	Stack	Terminals	Action
0			NT(S)
1	(S		NT(NP)
2	(S | (NP		GEN(*The*)
3	(S | (NP | *The*	*The*	GEN(*hungry*)
4	(S | (NP | *The* | *hungry*	*The* | *hungry*	GEN(*cat*)
5	(S | (NP | *The* | *hungry* | *cat*	*The* | *hungry* | *cat*	REDUCE
6	(S | (NP *The hungry cat*)	*The* | *hungry* | *cat*	NT(VP)
7	(S | (NP *The hungry cat*) | (VP	*The* | *hungry* | *cat*	GEN(*meows*)
8	(S | (NP *The hungry cat*) | (VP *meows*	*The* | *hungry* | *cat* | *meows*	REDUCE
9	(S | (NP *The hungry cat*) | (VP *meows*)	*The* | *hungry* | *cat* | *meows*	GEN(.)
10	(S | (NP *The hungry cat*) | (VP *meows*) | .	*The* | *hungry* | *cat* | *meows* | .	REDUCE
11	(S (NP *The hungry cat*) (VP *meows*) .)	*The* | *hungry* | *cat* | *meows* | .	

可以看出，generative model和discriminative model的基本结构相同，不同之处在于两点：第一，生成模型的buffer是逐渐向前的，而判别模型中buffer先卷一遍，之后慢慢从后先前输出；第二，在预测得到SHIFT操作时，在生成模型中被认为是GEN操作，需要用class-factored softmax生成词。

代码的主体流程基本和判别模型一致，这里就不在赘述了。

4. Reference

[1] Dyer C, Kuncoro A, Ballesteros M, et al. Recurrent Neural Network Grammars[J]. 2016
[2] https://github.com/clab/rnng
[3] https://github.com/clab/cnn-v1
[4]http://cslt.riit.tsinghua.edu.cn/mediawiki/images/2/2b/RNNG_Code_Use_Guide_simplified.pdf