# 法律声明

- 本课件包括：演示文稿，示例，代码，题库，视频和声音等，小象学院拥有完全知识产权的权利；只限于善意学习者在本课程使用，不得在课程范围外向任何第三方散播。任何其他人或机构不得盗版、复制、仿造其中的创意，我们将保留一切通过法律手段追究违反者的权利。

- 课程详情请咨询
  - 微信公众号：小象
  - 新浪微博：ChinaHadoop

# 神经序列模型 III

主讲人： 史兴

07/12/2017

# 有关提问

- 关于debug的问题
  - 欢迎在小象学堂上提问
  - 稀疏性
    - 能得到正确的回答往往是碰运气
    - 自己问的问题，自己最后把答案写上，减小稀疏性
    - stackoverflow 一般能查到这种问题

# 超参数搜索

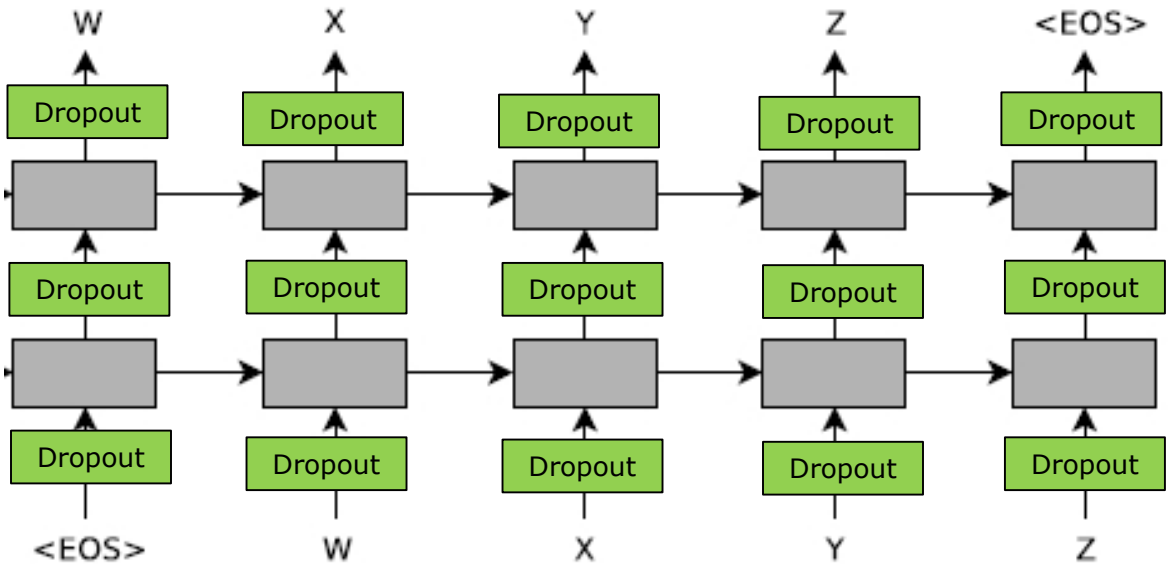
- 排名
  - http://collabedit.com/qvpc6
  - 第一名(13人参加)：xinfeng 1007
  - 神秘大奖：联系我领取
  - 比赛继续：
    - 我们下周开课前再看一下排行榜

# 提纲

- RNNLM代码讲解

# RNN代码讲解

- Github地址
  - https://github.com/shixing/xing_nlp/tree/master/LM/RNNLM
  - 任务： RNNLM



# RNN代码讲解

- 目录概况：
  - /data 存放数据
  - /model 存放训练好的模型
  - /py 所有的tensorflow的代码
  - /sh 调用我们的代码的bash script
  - /readme.md “项目的遗书”

# RNN代码讲解

- 目录概况：
  - /data 存放数据
    - /ptb (Penn-Tree bank)
      - train
      - valid
      - test
    - /small (随机生成的一些小数据)
      - train
      - valid
      - test

# RNN代码讲解

- 数据概览
  - 多少句(行)?
    - wc train
  - 多少个单词(token)?
    - wc train
  - 多少种单词(type)?
    - cat train|tr ‘ ’ ‘\n’|sort|uniq|wc
  - 最长(短)的句子有多少单词?
    - awk ‘{print NF}’|sort –n|uniq|head
  - 句子长度与数量的关系?
    - awk ‘{print NF}’|sort –n|uniq –c

# RNN代码讲解

- 目录概况：
  - /py 存放所有tensorflow相关的代码
    - best_buckets.py 计算最佳buckets配置
    - data_util.py 数据预处理
    - data_iterator.py 数据迭代器
    - run.py 训练、预测等入口
    - seqModel.py RNNLM模型的定义
    - generate_jobs.py 超参数grid search

# RNN代码讲解

- 数据预处理 data_util.py
  - 创建vocab
    - 添加特殊词汇
      - _PAD 填充词汇
      - _GO 句子开始
      - _EOS 句子结束
      - _UNK 未知词
    - xinfeng is number 1
    - _GO _UNK is number 1 _EOS
  - 将单词替换成数字
    - 1 3 102 3424 2

# RNN代码讲解

- 目录概况：
  - /data 存放数据
    - /ptb
      - train test valid
  - /model 存放训练好的模型
    - /model_ptb
      - /data_cache 预处理过的数据
        - train.id dev.id vocab
      - /saved_model
  - /py
  - /sh
  - /readme.md

# RNN代码讲解

- data_util.py
  - def prepare_data(cache_dir, train_path, dev_path, vocabulary_size)
    - create_vocabulary(vocab_path, [train_path, dev_path], vocabulary_size)
      - line 142-144 根据频率选取vocab
    - data_to_token_ids(train_path, train_ids_path, vocab_path)
      - line 180 按空格分词 + _GO/_EOS + 替换_UNK
    - data_to_token_ids(dev_path, dev_ids_path, vocab_path)

# RNN代码讲解

- run.py
  - def main(_)
    - parsing_flags() 读取flags, 构建目录
    - def train() 训练
    - def force_decode() 预测
      - 输入：句子s
      - 输出：句子的概率 log P(s)

# RNN代码讲解

- Mini-batch Gradient Descent Mini-batch梯度下降法
  1. 初始化参数
     - $\Theta = \text{uniform(d)}$
  2. 随机抽取m个数据点$\mathrm{T_m} = \{(x_i, y_i)|i = k_1, \dots, k_m\}$,计算偏导数
     - $\nabla\Theta = \frac{\partial \mathrm{Obj}(\Theta, T_m)}{\partial \Theta} = \frac{\partial}{\partial \Theta}\sum_{i=k_1}^{k_m} Obj(\Theta, x_i, y_i)$
  3. 更新参数
     - $\Theta = \Theta - \eta\,\nabla\Theta$
  4. 适当的条件更新learning rate $\eta$，返回2，直到收敛

# RNN代码讲解

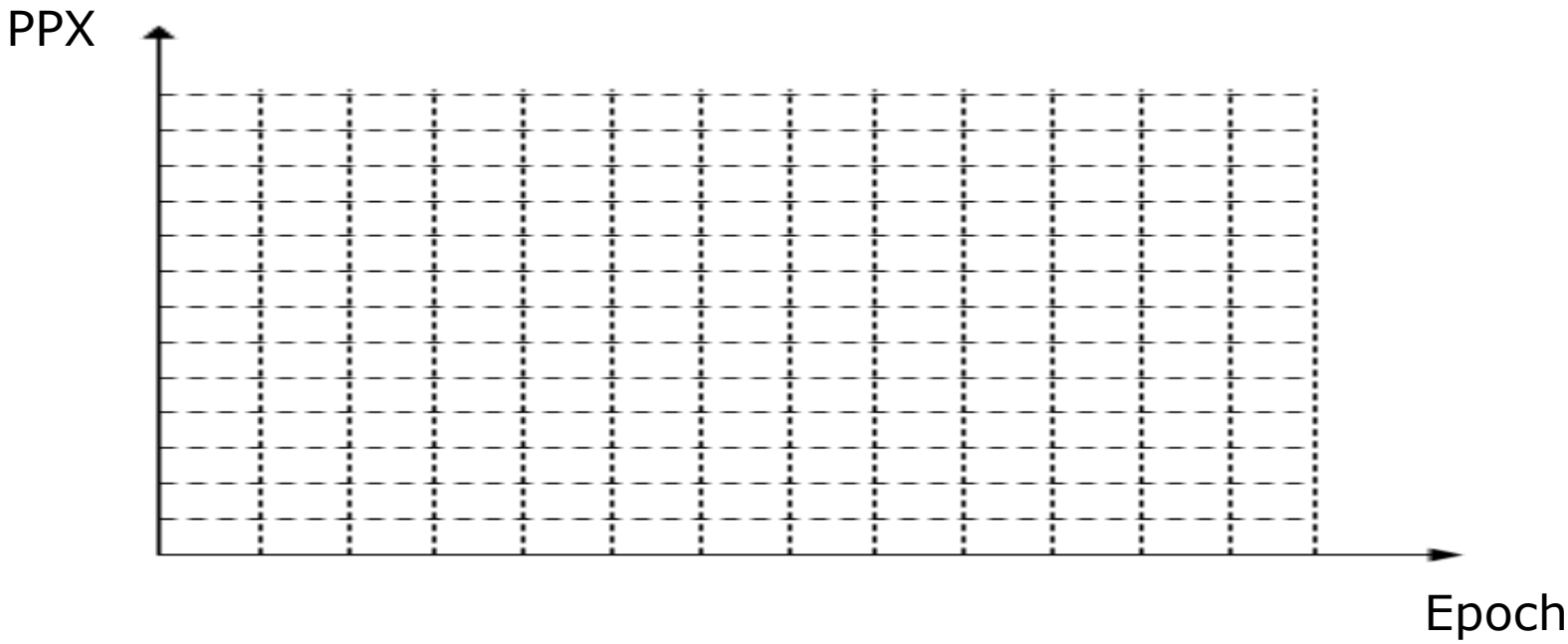
- Mini-batch Gradient Descent Mini-batch梯度下降法
  - 适当的条件更新learning rate $\eta$，返回2，直到收敛
    - 适当的条件:
      - 每处理了一半的训练数据，就去验证集计算perplexity
        - 如果perplexity比上次下降了，保持learning rate不变，记录下现在最好的参数
        - 否则， learning rate *= 0.5 缩小一半
      - 如果连续10次learning rate没有变，就停止训练
  - 蓝色的都是超参数

# RNN代码讲解

- Mini-batch Gradient Descent Mini-batch梯度下降法
  - 适当的条件更新learning rate $\eta$，返回2，直到收敛
    - 适当的条件:
      - 每处理了一半的训练数据，就去验证集计算perplexity
        - 如果perplexity比上次下降了，保持learning rate不变，记录下现在最好的参数
        - 否则， learning rate *= 0.5 缩小一半
      - 如果连续10次learning rate没有变，就停止训练
  - 蓝色的都是超参数

# RNN代码讲解

- Mini-batch Gradient Descent Mini-batch梯度下降法
  - 适当的条件更新learning rate $\eta$ ...

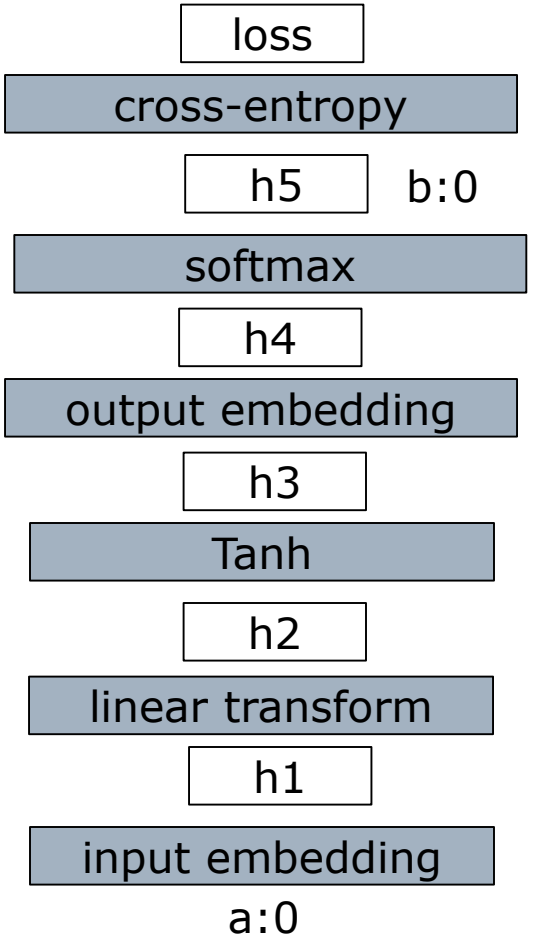


# RNN代码讲解

- run.py
  - def train()
    - 读取训练数据 train和验证数据dev
    - 建立模型; patience = 0
    - while
      - 从数据中随机取m个句子进行训练
      - 到达半个epoch，计算ppx(dev)
        - 比之前降低：更新best parameters
        - 比之前升高：learning rate 减半，patience +=1
      - if (patience>10): break

# RNN代码讲解

mini-batch = 1




h3
[ 0.07982977  0.7530659 ]
h2
[ 0.08  0.98]

linear_w
[[ 1.2  0.2]
 [-0.4  0.4]]

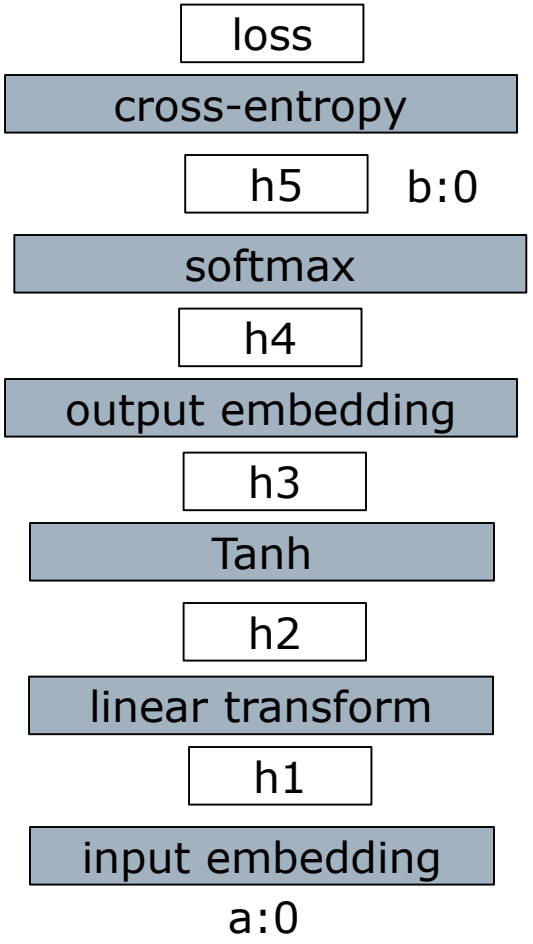
linear_b
[ 0.  0.5]

h1
[ 0.4  1. ]

input_embed
[[ 0.4  1. ]
 [ 0.2  0.4]
 [-0.3  2. ]]

# RNN代码讲解

mini-batch = 1




ce
0.810028205586
h5
[ 0.351601 0.444846 0.203553]

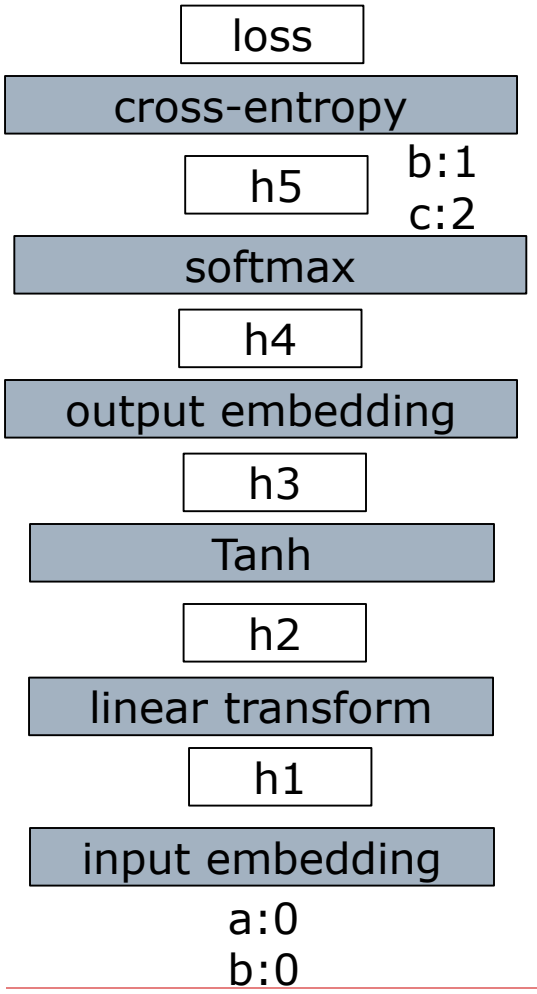
h4
[ 0.673236 0.908465 0.12666425]

output_embed
[[-1. 1. ]
[ 0.4 0.5]
[-0.3 0.2]]

output_embed_b
[ 0. 0.5 0. ]

# RNN代码讲解

mini-batch = 2




h3
[[ 0.07982977  0.7530659 ]
[ 0.07982977  0.60436778]]
h2
[[ 0.08  0.98]
[ 0.08  0.7 ]]

linear_w
[[ 1.2  0.2]
 [-0.4  0.4]]

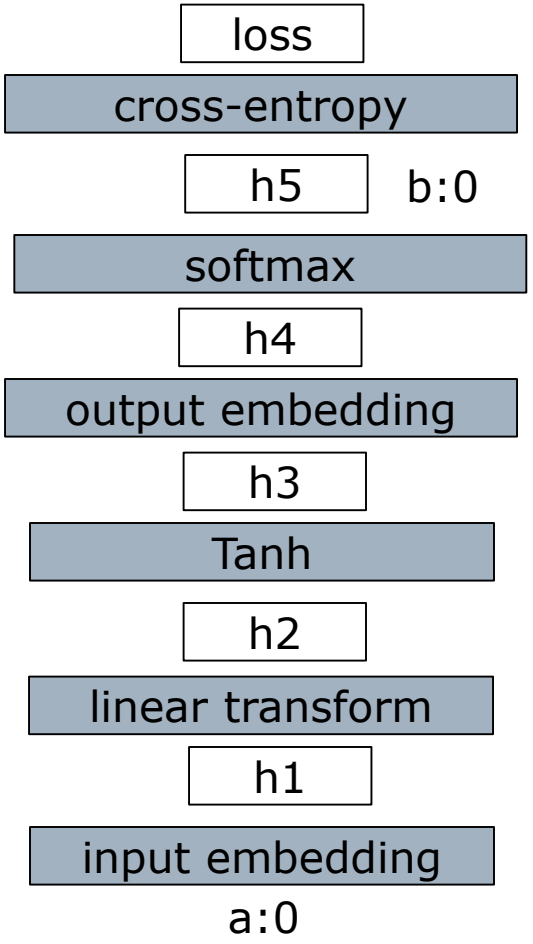
linear_b
[ 0.  0.5]

h1
[[ 0.4  1. ]
 [ 0.2  4. ]]

input_embed
[[ 0.4  1. ]
 [ 0.2  0.4]
 [-0.3  2. ]]

# RNN代码讲解

mini-batch = 2




ce
0.810028 + 1.53118 = 2.3412
h5
[[ 0.351601 0.444846 0.203553]
[ 0.331685 0.452036 0.216279]]

h4
[[ 0.673236 0.908465 0.12666425]
[ 0.524538 0.83412 0.096925]]

output_embed
[[-1. 1. ]
[ 0.4 0.5]
[-0.3 0.2]]

output_embed_b
[ 0. 0.5 0. ]

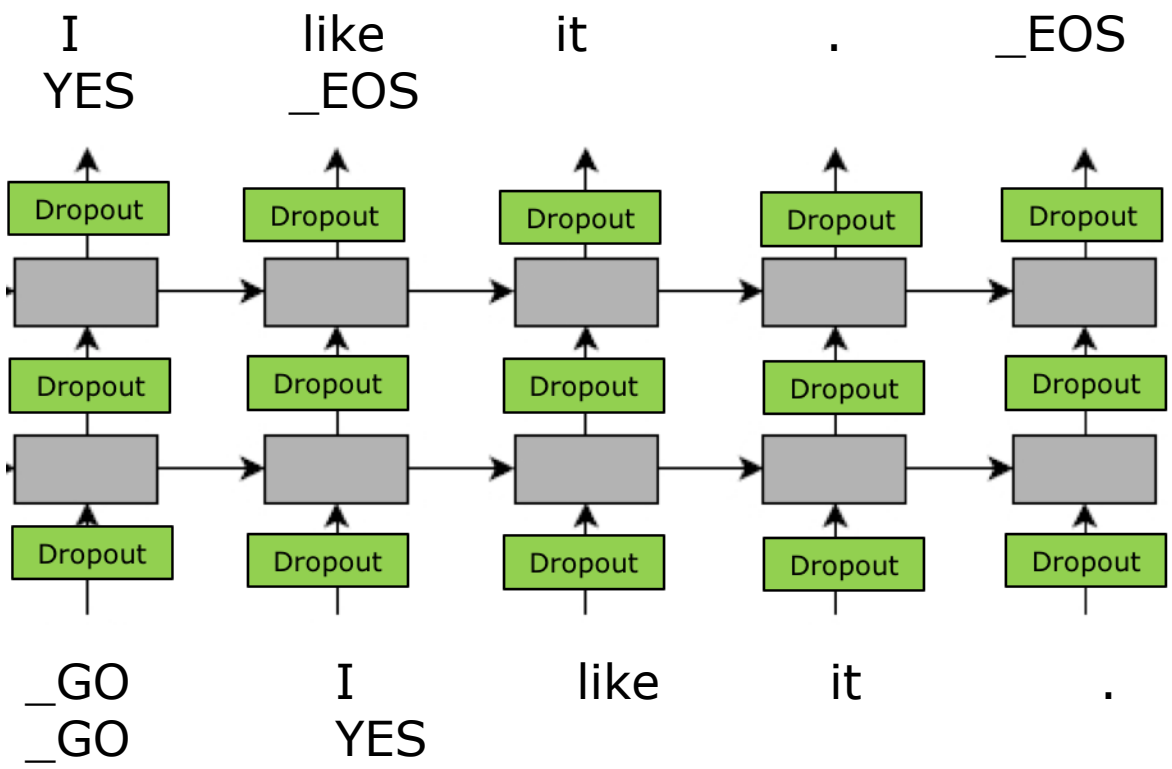
# RNN代码讲解

- mini-batch的优点：
  - 参数更新速度比Batch GD快速
  - 比SGD稳定
  - Matrix*Vector → Matrix*Matrix
    - 对于GPU来说，加速幅度会很大
      - GPU喜欢比较大规模的计算

# RNN代码讲解

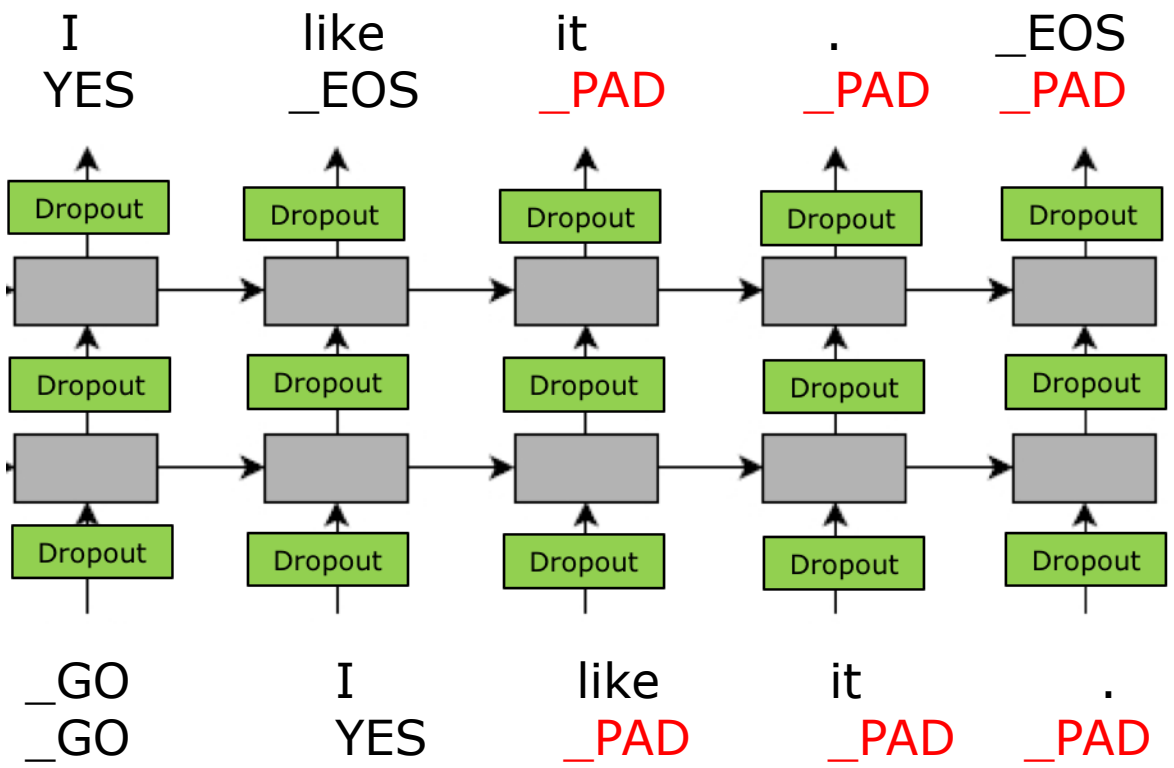
| 数量 | 句子长度 |
| --- | --- |
| 1101 | 10 |
| 1226 | 11 |
| … | … |
| 2 | 80 |
| 1 | 81 |
| 1 | 82 |

- mini-batch在RNN上问题
  - 句子的长度不一样



# RNN代码讲解

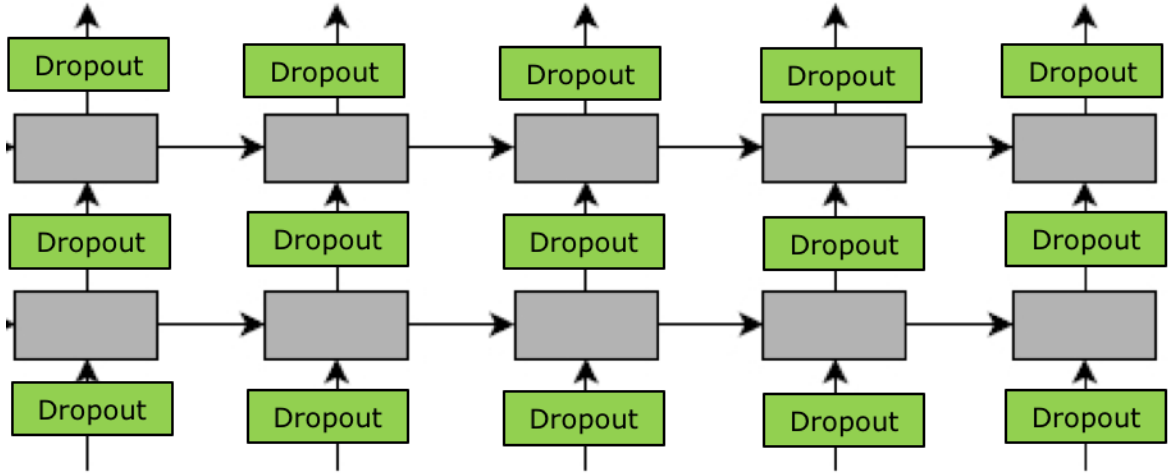
- mini-batch在RNN上问题
  - 句子的长度不一样： 增加padding



# RNN代码讲解

- mini-batch在RNN上问题
  - loss 增大了！

logP(I) + logP(like) + logP(it)+logP(.)+logP(_EOS) = Loss
+logP(YES)+logP(_EOS)+logP(_PAD)+logP(_PAD)+logP(_PAD)



# RNN代码讲解

- mini-batch在RNN上问题
  - loss 增大了！ 乘以一个0/1 mask

LOSS =
[[logP(I), logP(like), logP(it), logP(.), logP(_EOS)],
[logP(YES),logP(_EOS),logP(_PAD),logP(_PAD),logP(_PAD)]]
*
[[1,1,1,1,1],
 [1,1,0,0,0]]

= logP(I) + logP(like) + logP(it)+logP(.)+logP(_EOS)
+logP(YES)+logP(_EOS)

# RNN代码讲解

| 数量 | 句子长度 |
| --- | --- |
| 1101 | 10 |
| 1226 | 11 |
| 1 | 81 |
| 1 | 82 |

- mini-batch在RNN上问题
  - 增加Padding
  - loss 需要乘以mask
  - 效率降低了：
    - 所有句子补齐到82个字
      - 实际计算了(1101++1226+1+1) * 82 = 190978 步
      - 有效的步数：1101*10 +1226 * 11 + 1* 81+ 1*82 = 24659
      - 利用率： 12.9% 浪费！

# RNN代码讲解

| 数量 | 句子长度 |
| --- | --- |
| 1101 | 10 |
| 1226 | 11 |
| 1 | 81 |
| 1 | 82 |

- mini-batch在RNN上问题
  - 效率降低了：
    - 所有句子补齐到82个字
      - 实际计算了(1101++1226+1+1) * 82 = 190978 步
      - 有效的步数：1101*10 +1226 * 11 + 1* 81+ 1*82 = 24659
      - 利用率： 12.9% 浪费！
    - 提升效率
      - 将句子分成两组， 一组补齐到11，一组补齐到82
        - (1101+1226) * 11 + (1+1)*82 = 25761
        - 利用率： 24659 / 25761 = 95.7%

# RNN代码讲解

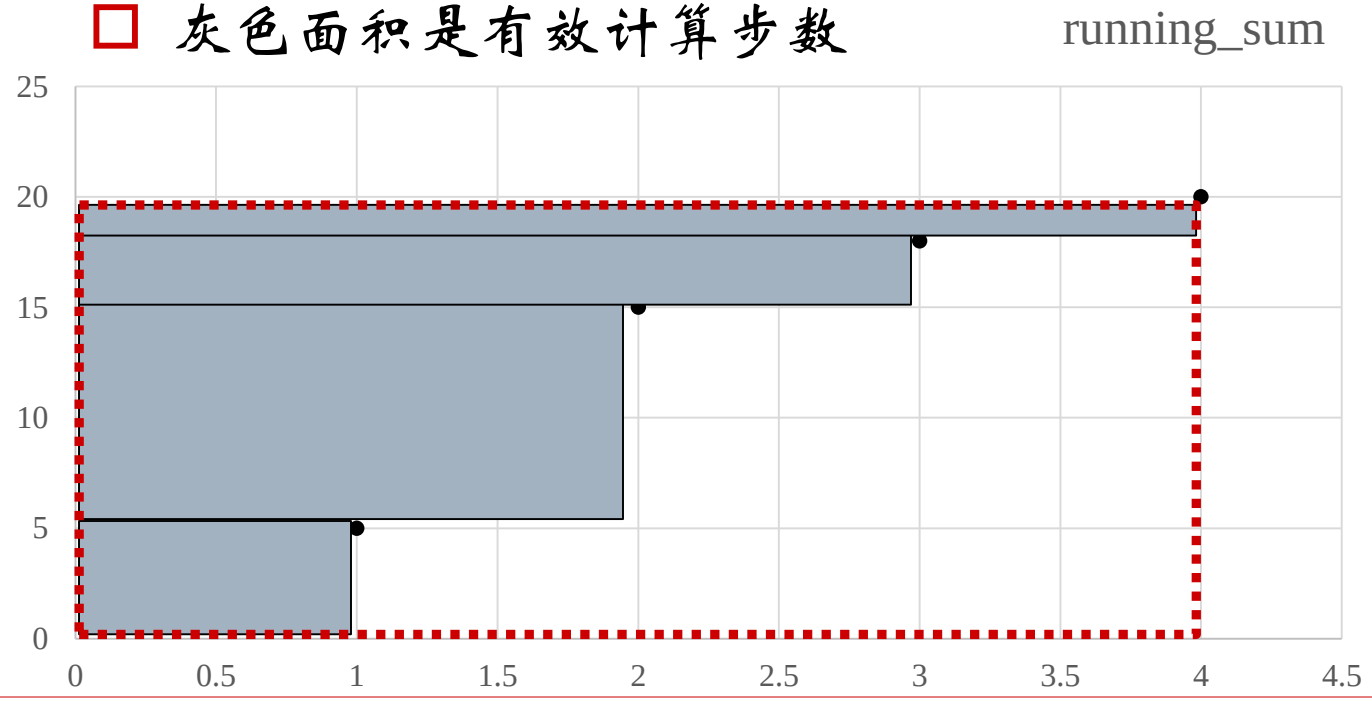
- best_buckets.py
  - 增加Padding
  - loss 需要乘以mask
  - 将句子分组
    - 如何分组？
    - 如何达到最优的分组？
    - best_buckets.py

# RNN代码讲解

- bucket.py
  - def calculate_buckets(length_array, max_length, max_buckets)
  - 算法：贪心，二分类
    - length_array = [1,1,1,1,1,2,2,2,2,2,2,2,2,2,2,3,3,3,4,4]
    - max_buckets = 3
    - max_length = 4
    - running_sum = [(1,5),(2,15),(3,18),(4,20)]

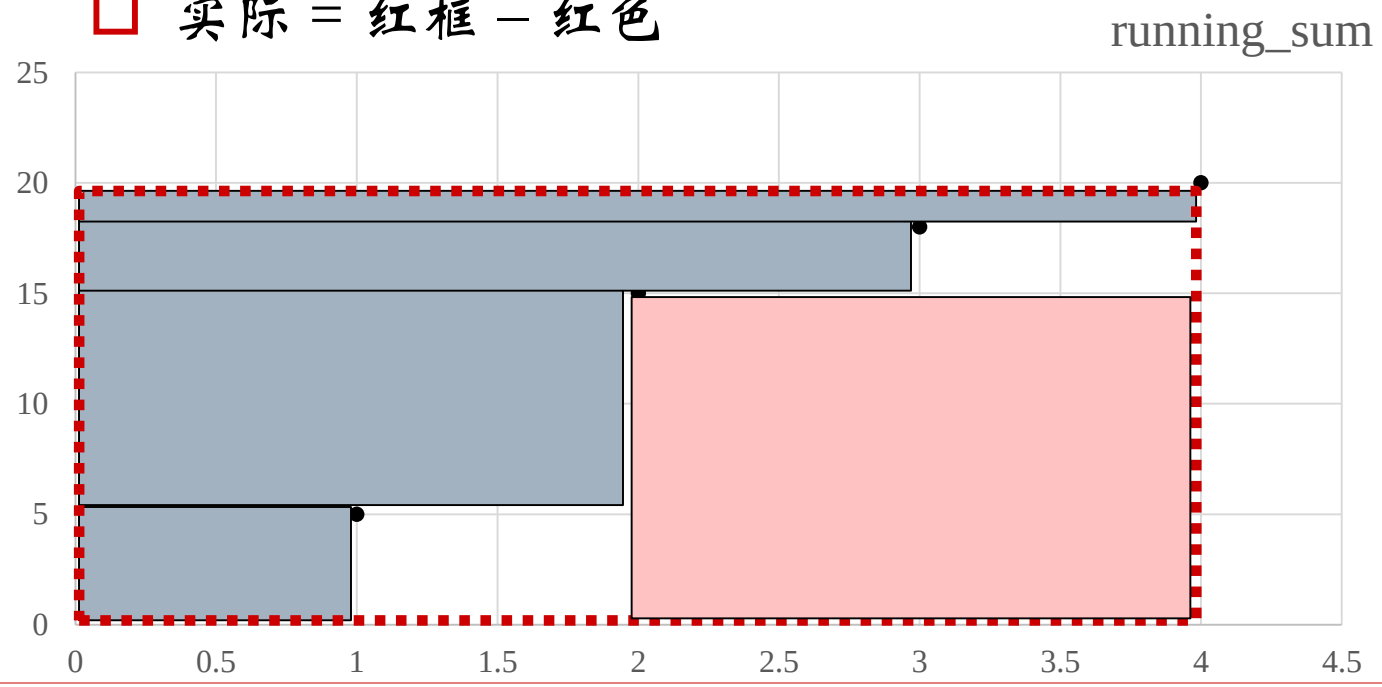
# RNN代码讲解

- bucket.py
  - 算法：贪心，二分类
    - running_sum = [(1,5),(2,15),(3,18),(4,20)]
    - 灰色面积是有效计算步数



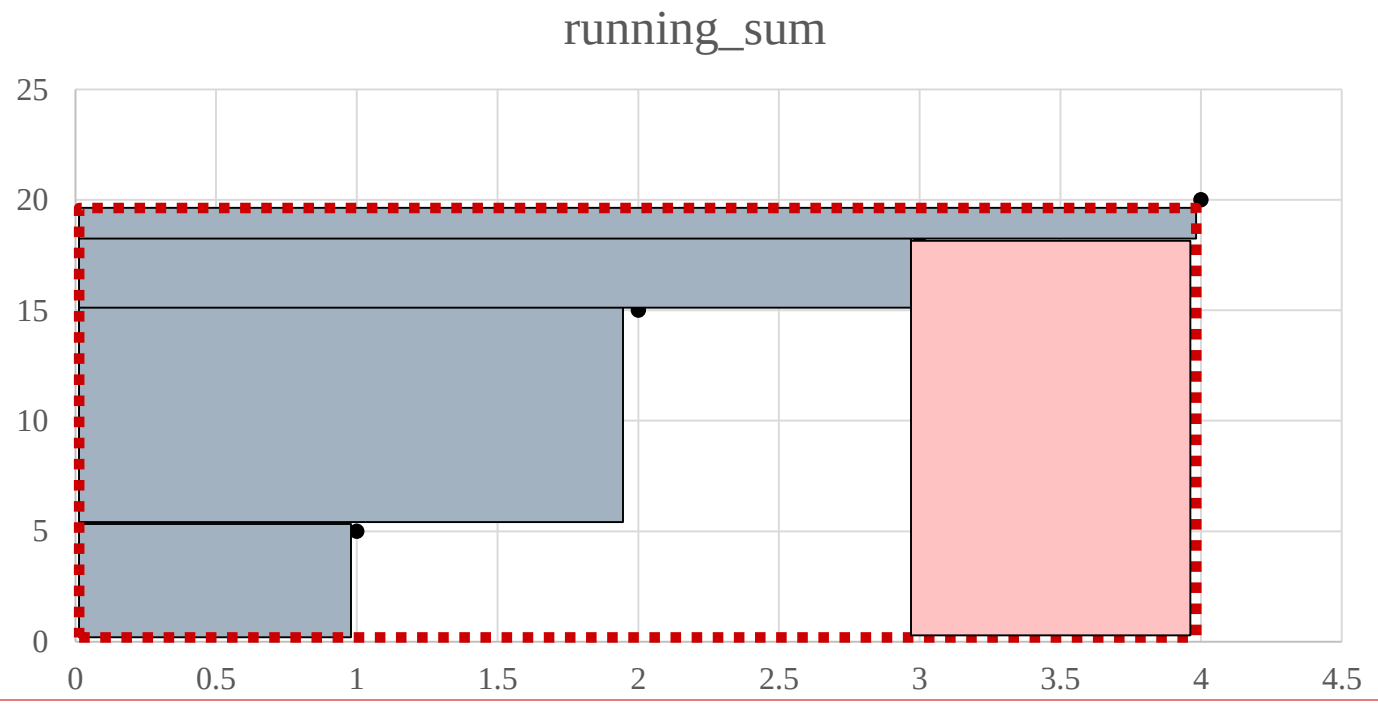
# RNN代码讲解

- bucket.py
  - 算法：贪心，二分类
    - 如果buckets = [2,4];
    - 实际 = 红框 – 红色



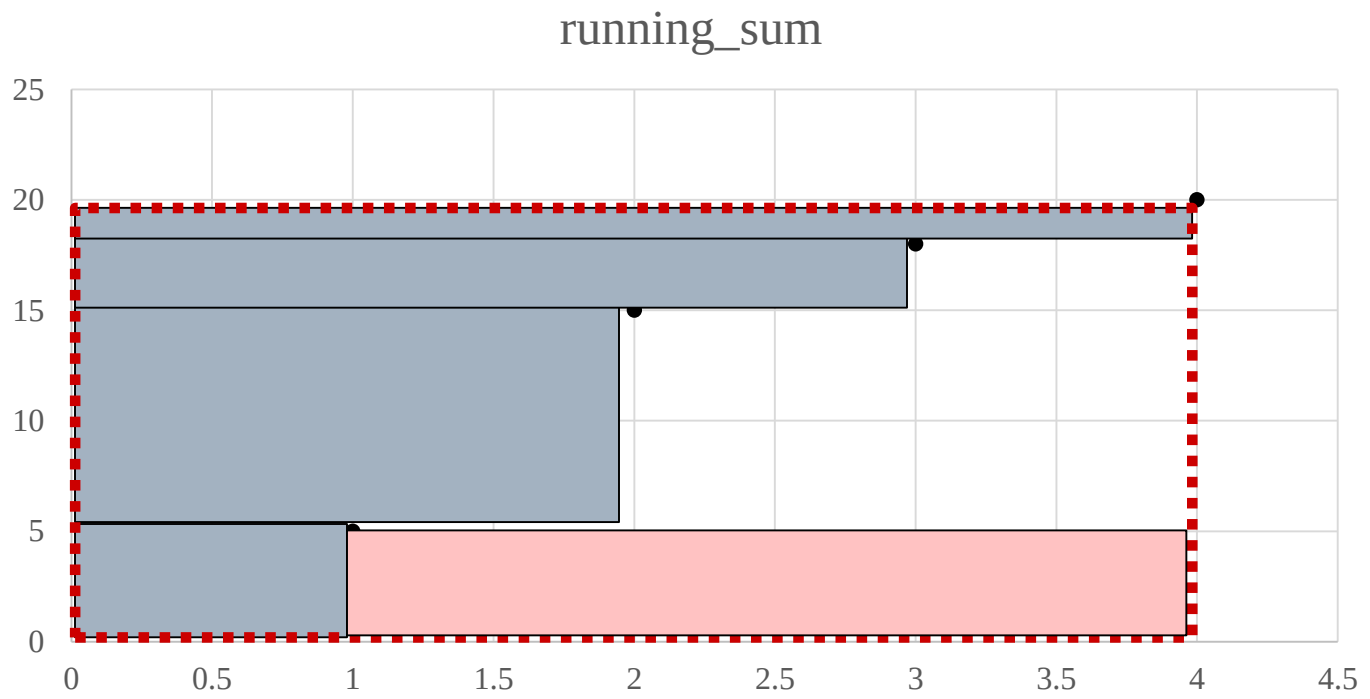
# RNN代码讲解

- bucket.py
  - 算法：贪心，二分类
    - 如果buckets = [3,4]



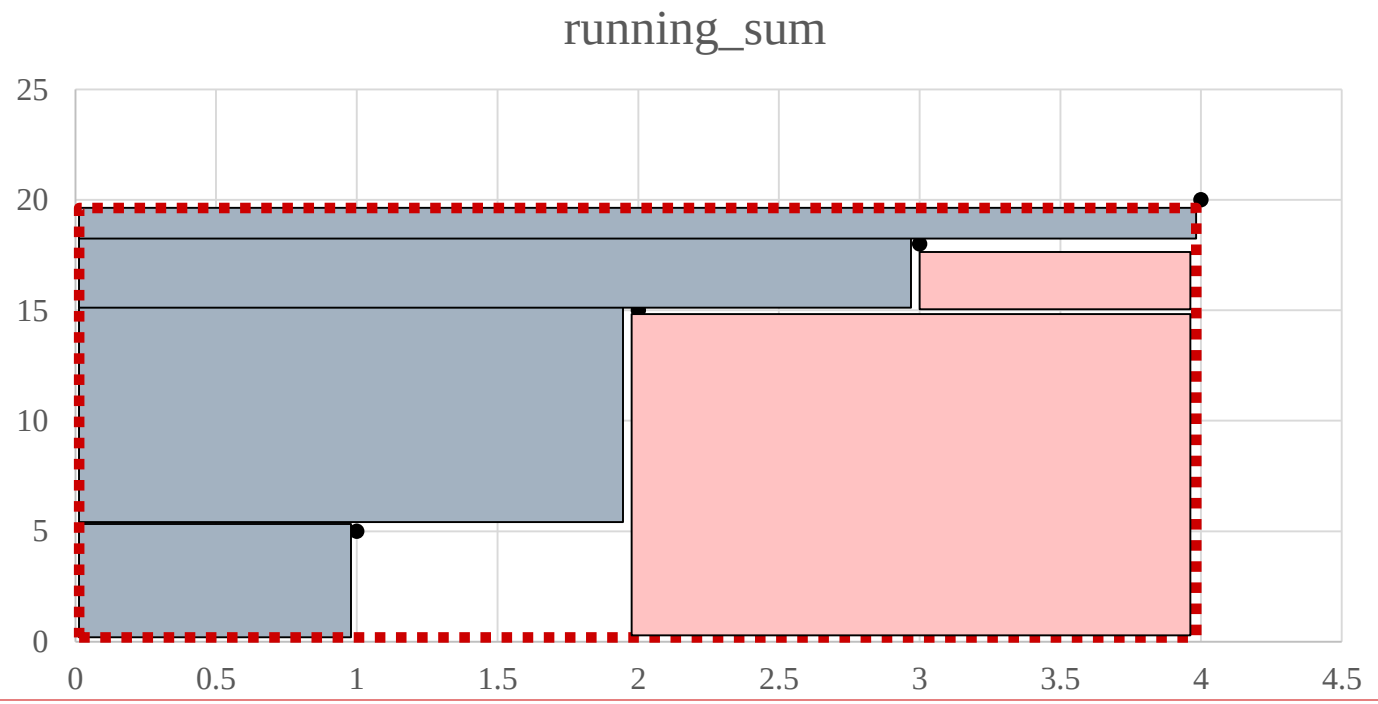
# RNN代码讲解

- bucket.py
  - 算法：贪心，二分类
    - 如果buckets = [1,4]



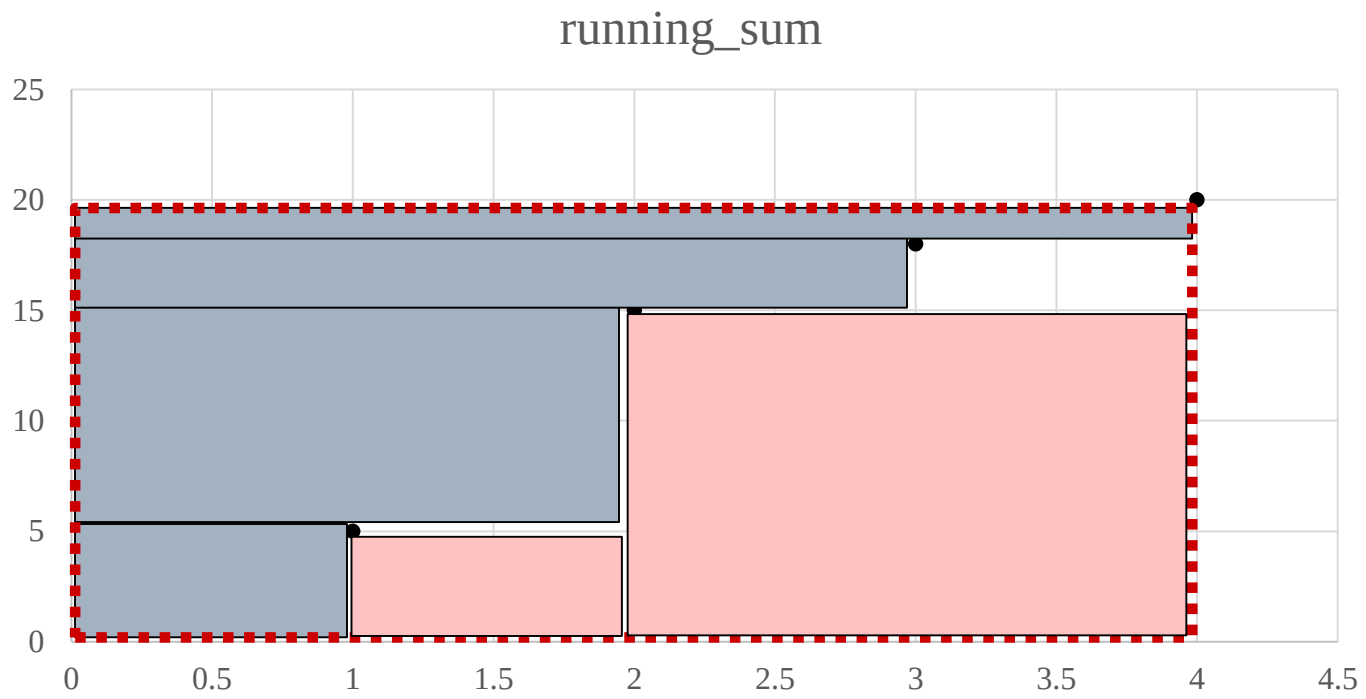
# RNN代码讲解

- bucket.py
  - 算法：贪心，二分类
    - buckets = [2,4,3]



# RNN代码讲解

- bucket.py
  - 算法：贪心，二分类
    - buckets = [2,4,1]；最好的buckets = [1,2,4]



# RNN代码讲解

- buckets对数据的影响
  - line 177: train_data_bucket, dev_data_bucket, _buckets, vocab_path = read_train_dev(FLAGS.data_cache_dir, FLAGS.train_path, FLAGS.dev_path, FLAGS.vocab_size, FLAGS.L, FLAGS.n_bucket)
  - _buckets = [1,2,4]
  - train_data_buckets = [ [ 1,1,1,1,1] , [2,2,2,2,2,2,2,2,2,2], [3,3,3,4,4] ]

# RNN代码讲解

- buckets对数据的影响
  - train_data_buckets = [ [ s1,s1,s1,s1,s1] , [s2,s2,s2,s2,s2,s2,s2,s2,s2,s2], [s3,s3,s3,s4,s4] ]
  - 如何随机选择m个数据？(data_iterator.py)
    - def next_random(self)
      - inputs, outputs, weights, _ = self.model.get_batch(self.data_set, bucket_id)
      - 先随机一个buckets | 第三个bucket
      - 再随机取m个数据 | m = 2, [s3, s4]
      - 将m个数据变成一个矩阵，加上padding

# RNN代码讲解

- buckets对数据的影响
  - 如何随机选择m个数据？(data_iterator.py)
    - def next_random(self)
      - inputs, outputs, weights, _ = self.model.get_batch(self.data_set, bucket_id)
      - 先随机一个buckets | 第三个bucket
      - 再随机取m个数据 | m = 2, [s3, s4]
      - 将m个数据变成一个矩阵，加上padding
      - s3 = [1,232,2], s4 = [1, 233,93,2]
      - inputs = [[1,232,0,0], [1,233,93,0]]
      - outputs = [[232,2,0,0], [233,93,2,0]]
      - weights = [[1,1,0,0],[1,1,1,0]]

# RNN代码讲解

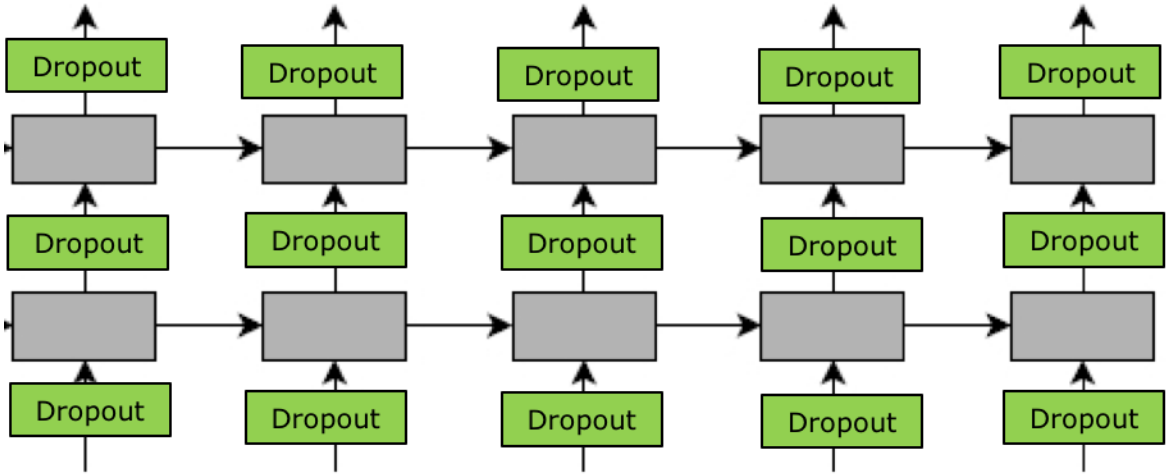
- run.py 过一遍代码
  - def train()
    - 读取训练数据 train和验证数据dev
    - 建立模型; patience = 0
    - while
      - 从数据中随机取m个句子进行训练
      - 到达半个epoch，计算ppx(dev)
        - 比之前降低：更新best parameters
        - 比之前升高：learning rate 减半，patience +=1
      - if (patience>10): break

# RNN代码讲解

- seqModel.py
  - def __init__() 建立模型
  - def get_batch(self, data_set, bucket_id, start_id = None)
    - 给定bucket_id, 生成相应的inputs, outputs, weights.
  - def step(self,session, inputs, targets, target_weights, bucket_id, forward_only = False, dump_lstm = False)
    - forward + backward + weights_updates;

# RNN代码讲解

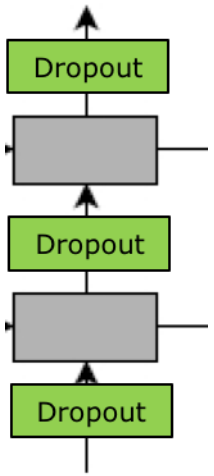
- def __init__() 建立模型
  - Input Layer




self.inputs_embed = [[0.4,0.3],[1,0.3], [0.4,-0.3],[1.3,0.3], [0,0.3],]
self.inputs = [1 43 34 23 0 ]

# RNN代码讲解

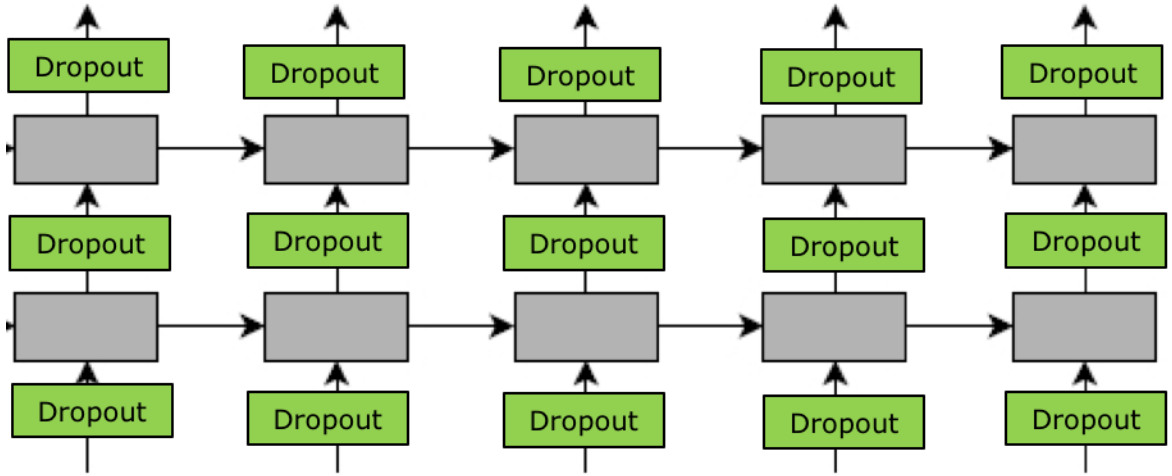
- def __init__() 建立模型
  - LSTM (line 103 - 111)
  - single_cell =



# RNN代码讲解

- def __init__() 建立模型
  - Output Layer

self.target_weights = [1  1  1  1  1]
self.targets = [ 43  34  23  2  0 ]

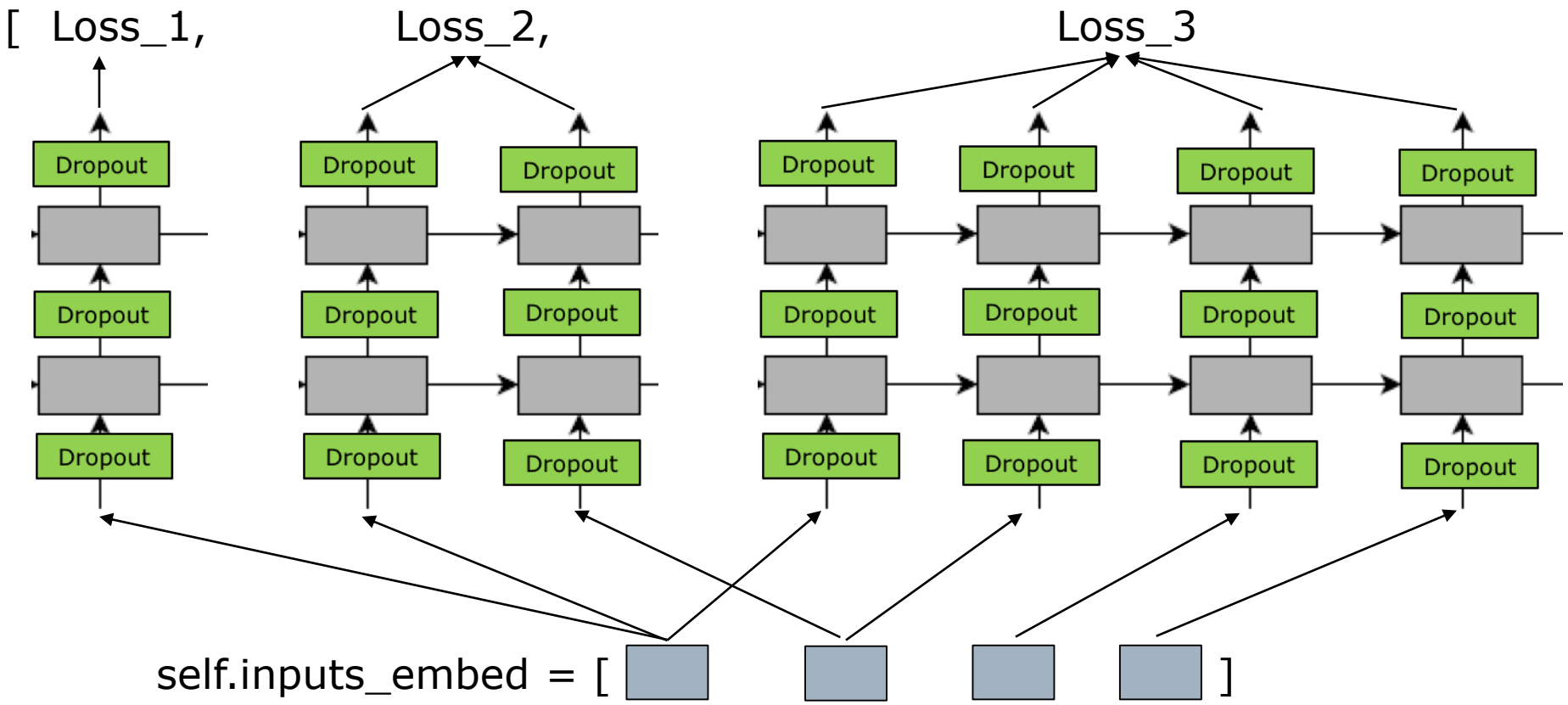


# RNN代码讲解

- buckets对模型的影响
  - self.model_with_buckets(…)
  - buckets = [1,2,4]

# RNN代码讲解

self.losses =



# RNN代码讲解

- def __init__() 建立模型
  - #Train
  - gradients = tf.gradients(self.losses[b], params, colocate_gradients_with_ops=True)
  - self.updates 更新参数的operation

# RNN代码讲解

- 快速过一遍：
  - def get_batch()
  - def step()

# 联系我们

## 小象学院：互联网新技术在线教育领航者

- 微信公众号：大数据分析挖掘
- 新浪微博：ChinaHadoop