**Self-critical程序解读**（作者：李舒燕）

**Dataset：**

coco val／train2014文件夹中的images， dataset\_coco.json中的图片-caption信息

**数据预处理：**

包含scripts文件夹下三个py文件。

Prepro\_feats.py：得到图片经过CNN的feature，CNN可选择resnet101或resnet152. 待处理数据集为raw\_images和dataset\_coco.json，输出的feature存放在fc和att两个文件夹下（每个文件夹下根据不同的图片有一堆文件），对应全连阶层和卷积层的feature。

Prepro\_labels.py：待处理数据集为dataset\_coco.json，包含两个子函数build\_vocab和encode\_captions，前者生成词典，返回的vocab是一个list，筛选掉了出现次数小于门限值的word，假如有word被筛掉，则在vocab最后append上一个'UNK'标记。注意这个vocab没有放入起始、终止符号。程序中门限值取5. 根据vocab得到wtoi和itow，注意i是从1开始的。后者限定了句子的max\_length（默认值为16），句子长度一旦长于这个限定就截掉后面的部分，根据wtoi对句子里的词编码成index，不足最大长度的部分用0做padding。把所有caption的labels都concatenate到一起，得到的编码完的caption集L是一个num\_caption\*max\_length的矩阵。label\_start\_ix[i] 和label\_end\_ix[i]记录第i个image对应的起始caption和终止caption的index。label\_length记录每个caption的长度。输出有两个文件，h5文件包含有编码完的caption、label\_start\_ix 和label\_end\_ix、label\_length信息。Json文件中包含itow，split，cocoid和图片文件路径信息。

Prepro\_ngrams.py：输出为用于self-critical阶段计算cider的中间结果。

**训练：**

train.py:用dataloader.py加载数据，通过data = loader.get\_batch('train')得到一个batch的训练数据（feature，labels，mask）。注意在dataloader中将labels的前面都padding上了一个0作为start标记。判断是否加载训练过的网络，在start from路径下有model时将model加载进来，并从info文件中读当前的epoch，iteration，optimazer等状态。重新训练网络时，对epoch，iteration等初始化为0。设置模型，fc和attention模型可选。设置学习率的decay，是否使用schedule sample，是否在某个epoch以后进行self-critical（在这个特定的epoch之前用传统的交叉熵模型）。

FCModel.py: 两个类LSTMCore和FCModel，前者是自己定义的一个LSTM计算网络，输出为ht和状态（ht, ct）。FCModel主要看forward和sample两个函数。Forward用于训练网络，i=0时刻输入feature，假如不使用schedule sample，后面时刻的输入依次为label从第一个到倒数第二个index。假如使用schedule sample，当i>=2时以ss\_prob的概率得到该batch中要用到ss的样本，按照i-1时刻网络输出的vocab上每一位的概率采样，得到一个index作为i时刻该样本输入的label，其余的样本仍以ground truth的label作为输入（这个采样在每一时刻都会做）。当这个batch上所有label都进入padding状态时即输入完成，break。（实验时没有用ss）

Sample用于测试网络，t=0时刻输入为feature，t=1时刻输入为0，之后输入的label为上一次输出的最大结果的位置。一旦出现0则代表该caption生成完毕，当一个batch的caption全生成完毕则break。输出为batch\*cap\_len的encoded句子和对应每个index的概率对数,其中cap\_len为该batch中最长的句子长度。

两个loss：在未启动self-critical前loss的计算在LanguageModelCriterion(nn.Module)中,用crossentropy对非padding位的logp求平均。

Self-critical的loss：首先用sample得到一个batch的encoded句子以及对应的概率。get\_self\_critical\_reward用于计算gen\_result和gready\_res的cider之差，前者直接用max得到index，后者依照概率进行采样得到index。用RewardCriterion得到最终的reward。

保存训练网络的参数：每个save\_checkpoint\_every更新网络参数到model.pth。在验证集上进行测试，用eval\_split得到验证集上的交叉熵loss，生成的句子以及有coco\_val得到的bleu，cider等评分。当验证集上的得分比之前最好的得分都高时保存网络参数到model-best.pth保存。这个得分在交叉熵训练阶段为验证集的-loss，在self-critical阶段为验证集的cider。同时保存info和history信息。

**测试：**

eval.py

和验证集上的测试没区别，还是用eval\_split函数。