

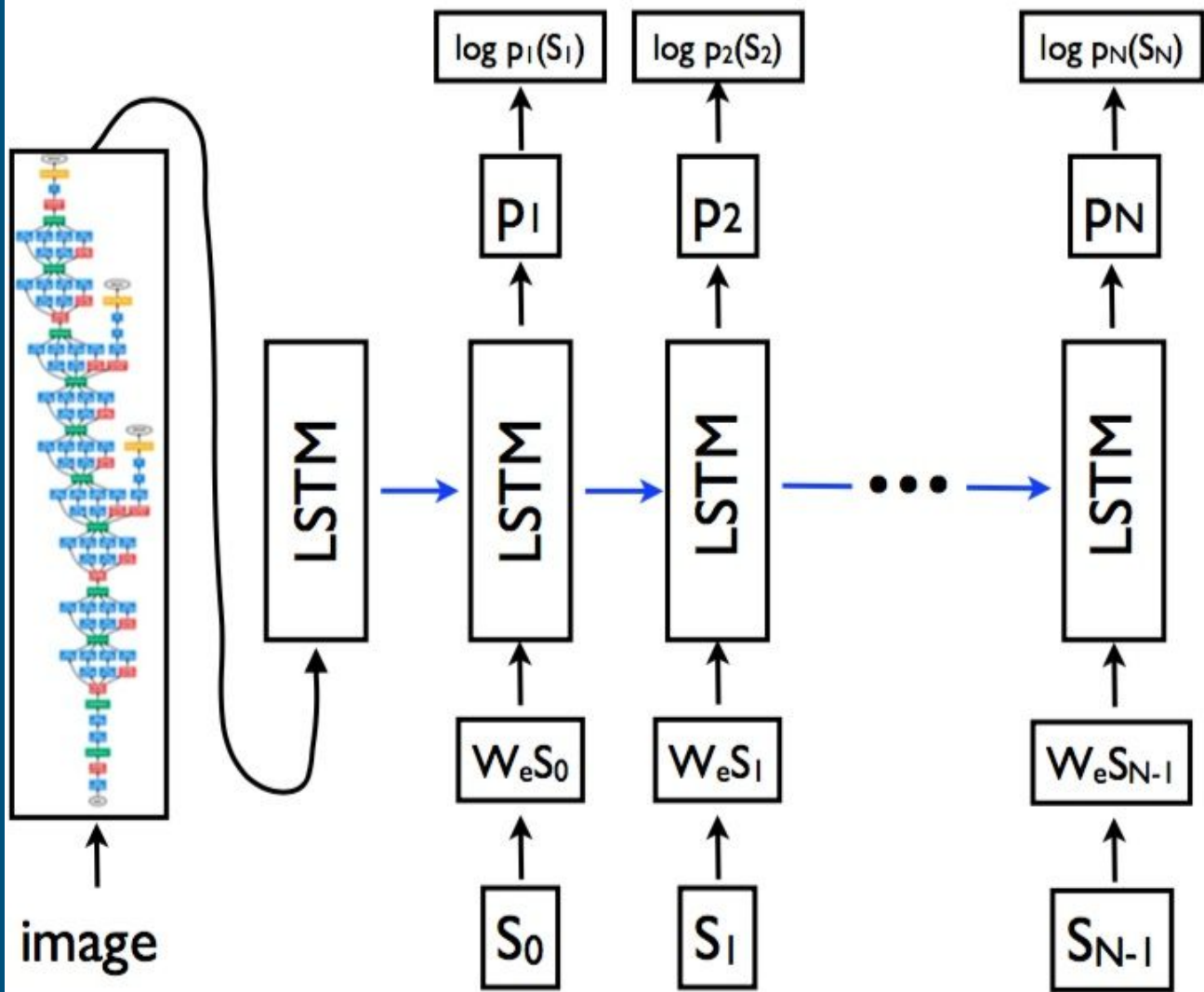
IM2TXT模型细节

经典模型架构

NIC(2014) : Show and Tell: A Neural Image Caption Generator

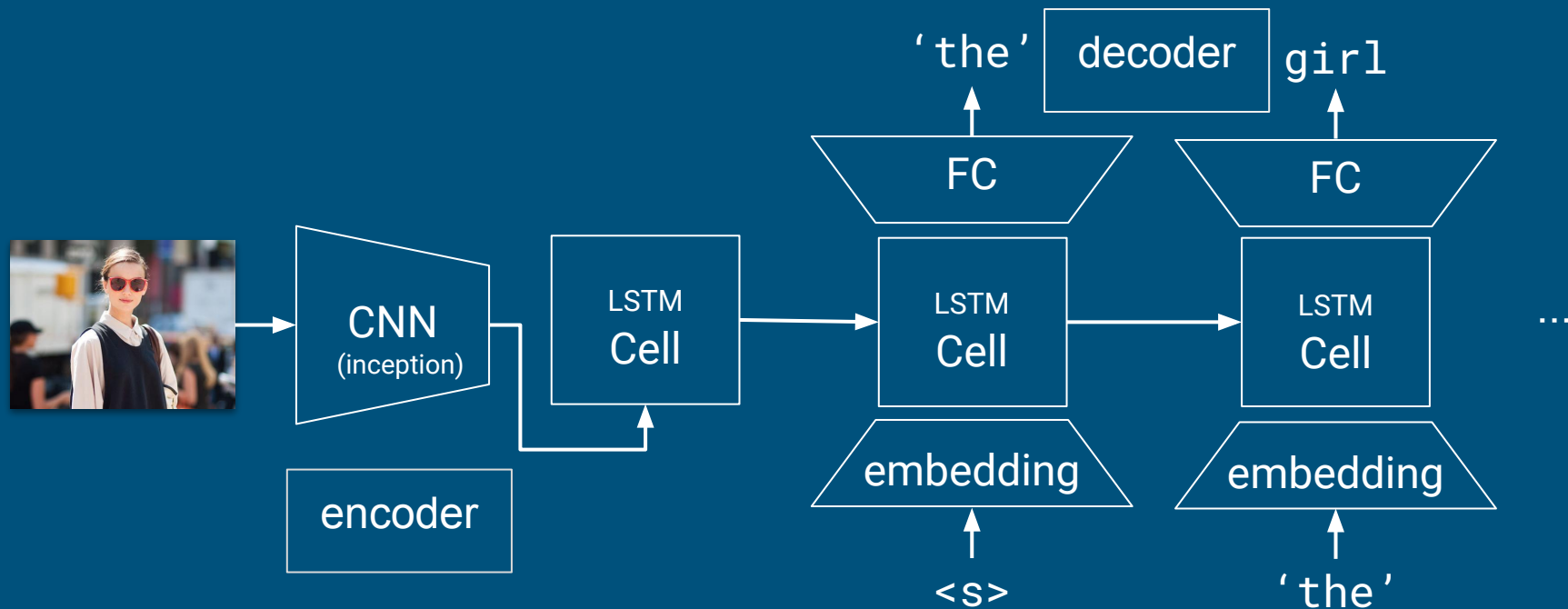
论文采用encoder-decoder结构, encoder部分用CNN网络提取到图片的特征向量, decoder基于前者的输出用LSTM逐个输出字符, 论文采用beam search来减小搜索空间获取最佳语句。

2015年Google团队携NIC参加MSCOCO Image Captioning Challenge ,赛后总结了一些经验并提出了对NIC的改进 (使用得分更高的特征提取网络,并直接载入预训练的网络,使用batch norm,提升LSTM层数和CELL深度,LSTM随机初始化,使用dropout,用更少的Beamsearch参数预测),另外他们也从实验中得出大语料训练出来的word embedding对模型作用极小,在不同数据集下模型得分差异较大;Google随后把代码开源到了Tensorflow下的Im2txt。



im2txt模型架构

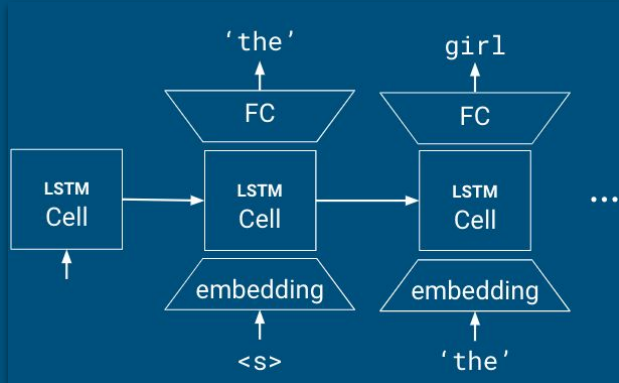
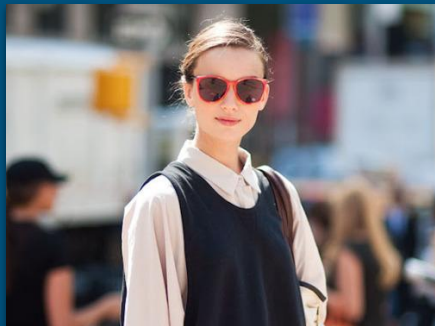
Show and Tell: Lessons learned from the 2015 MSCOCO Image Captioning Challenge[arxiv:1609.06647]



im2txt之模型细节

encoder结构中使用的了InceptionV3网络作特征提取的, 网络已经在LSVRC-2012-CLS图像分类数据集上进行了多轮训练, 这里只需加载预训练好的模型即可, 并且使用了batch normalization 提高网络训练的性能。

训练网络期间, CNN网络参数在一开始是固定不 进行训练, 在LSTM网络训练得到较平稳的loss曲线后, 取消CNN参数的固定, 让CNN与LSTM一起训练调参, 论文中提到这样做带来的效果是 颜色可以被正确描述出来。



decoder结构中的LSTM, 它的开始时刻的输入是在encoder中的提取的特征向量(8, 8, 2048), 它的常规输入是word embedding vectors, 每步的输出是词汇表中所有单词的概率。训练目标就是最大化全部 训练样本的对数似然之和。

$$\log P(S|I; \theta) = \sum_{t=0}^N \log P(S_t|S_0, S_1, \dots, S_{t-1}, I; \theta)$$

$$\theta^* = \arg \max_{\theta} \sum_{(I,S)} \log P(S|I; \theta)$$

im2txt之beamsearch

在seq2seq中获取decoder的输出结果, 常用方法有贪心搜索(Greedy Search)集束搜索(Beam Search)。贪心搜索只选择了概率最大的一个, 而集束搜索则选择了概率最大的前k个。这个k值也叫做集束宽度(Beam Width)。集束搜索本质上也是贪心的思想, 当k等于1的时候就是贪心搜索, 只不过它考虑了更多的候选搜索空间, 因此可以得到更多的输出结果。

在im2txt模型的测试生成过程中, 根据一张图片P, 推断联合概率最大的序列作为输出, 但要计算全部序列的概率然后选出概率值最大的序列这么做却不可行的, 因为每个位置都有整个词表规模的词作为候选, 搜索规模会随序列的长度而指数级增长, 所以这里就用到了beam search来缩小搜索空间, 获取最佳的输出语句。

google在[Show and Tell: Lessons learned from the 2015 MSCOCO Image Captioning Challenge](#) 中提到了对他们的beam width 取了从1至20范围内多组值进行训练, 后来发现值为3效果最佳, 所以在im2txt模型中caption_generator.py设置了beam_size=3 。

在我们的实验中对beam width 的值修改了1~20内的多组值, 发现值3的时候生成语句效果的确比较好。

TABLE 1

Scores on the MSCOCO development set for two models: NIC, which was the model which we developed in [46], and NICv2, which was the model after we tuned and refined our system for the MSCOCO competition.

Metric	BLEU-4	METEOR	CIDER
NIC	27.7	23.7	85.5
NICv2	32.1	25.7	99.8
Random	4.6	9.0	5.1
Nearest Neighbor	9.9	15.7	36.5
Human	21.7	25.2	85.4

TABLE 2

BLEU-1 scores. We only report previous work results when available. SOTA stands for the current state-of-the-art.

Approach	PASCAL (xfer)	Flickr 30k	Flickr 8k	SBU
Im2Text [18]	25	55	48	11
TreeTalk [14]				19
BabyTalk [3]				
Tri5Sem [16]				
m-RNN [27]				
MNLM [29] ⁵		56	51	
SOTA	25	56	58	19
NIC	59	66	63	28
Human	69	68	70	

im2txt之模型成果

数据摘自 Show and Tell [arxiv:1609.06647]

im2txt之输入数据处理

im2txt/data/build_mscoco_data.py

处理流程：根据字幕文件提取图片ID和对应语句 => 分配比例 => 建立词汇表 & 图片数据预处理 => 生成TFrecord
mscoco数据集分配如下：


















train_dataset = 100% of mscoco_train_dataset + 85% of mscoco_val_dataset.

val_dataset = 5% of mscoco_val_dataset (for validation during training).

test_dataset = 10% of mscoco_val_dataset (for final evaluation).

数据集中annotations下的json文件生成训练RNN需要的词汇表word_counts

生成的TFrecords文件包含256个训练集、4个验证集和8个测试集：

 train-00253-of-00256	 test-00000-of-00008
 train-00254-of-00256	 test-00001-of-00008
 train-00255-of-00256	 test-00002-of-00008
 val-00000-of-00004	 test-00003-of-00008
 val-00001-of-00004	 test-00004-of-00008
 val-00002-of-00004	 test-00005-of-00008
 val-00003-of-00004	 test-00006-of-00008
 word_counts.txt	 test-00007-of-00008
	 train-00000-of-00256

其中，每个tfrecord文件中包含约2300条记录，每条记录都是一个序列化的SequenceExample协议，其格式如下：

Context:

image/image_id: MSCOCO图像ID

image/data: 关于图像完整RGB信息的字符串

feature_lists:

image/caption: 包含带标记的字幕语句的字符串列表

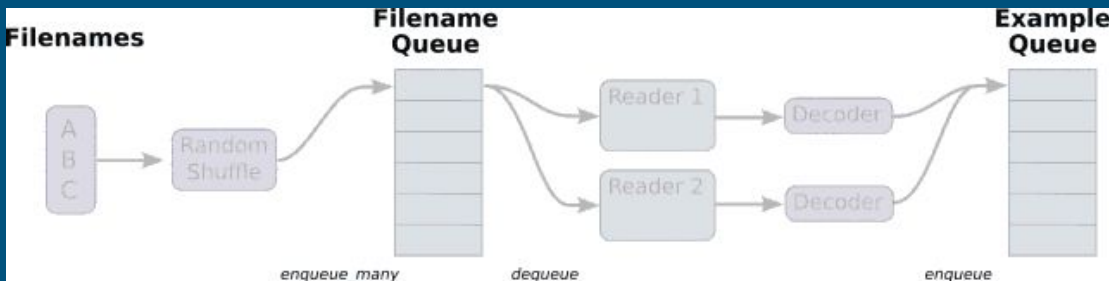
image/caption_ids: 与字幕语句对应的id列表

至于训练数据这么处理的好处：用硬盘空间换内存，减少内存开销，加载速度也更快；可以随意改组调用数据；数据在Tensorflow中异步预处理更加友好。

im2txt训练

对于训练数据的读取, im2txt在`show_and_tell_model.py`的`build_inputs`方法中调用了基于Tensorflow框架的文件队列功能, 实现了数据pipeline, 在模型训练过程中, tensorflow会按照队列内的文件自动调度, 前面我们进行预处理的TFrecords文件会被框架自动载入内存, 性能也很也非常不错。

这个流程分为两个阶段: 第一个阶段将输入文件打乱, 并在文件队列入列, 然后Reader从文件队列中读取一个文件, 同时文件队列出列这个文件, Reader同时对文件进行解码, 然后生产数据样本, 并将样本在样本队列中入列, 可以定义多个Reader并发地从多个文件同时读取数据。从样本队列中的出列一定量的样本数据即可以用于一个训练过程。如图:



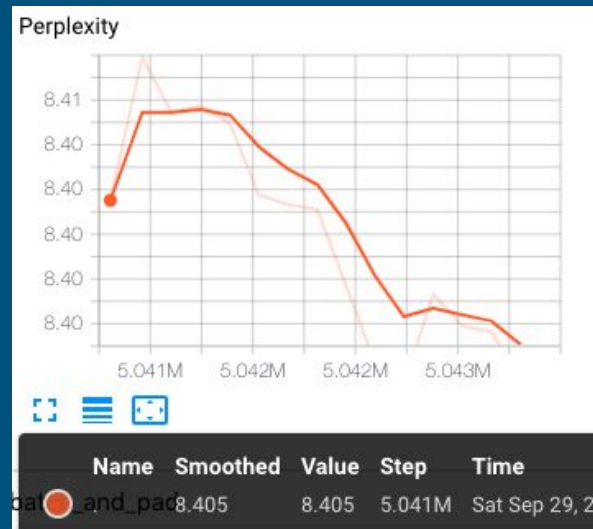
只对decoder部分的训练

```
INFO:tensorflow:Starting Queues.
INFO:tensorflow:global_step/sec: 0
INFO:tensorflow:global step 1: loss = 9.4074 (14.688 sec/step)
INFO:tensorflow:Recording summary at step 1.
INFO:tensorflow:global step 2: loss = 9.0141 (3.290 sec/step)
INFO:tensorflow:global step 3: loss = 7.5687 (0.950 sec/step)
INFO:tensorflow:global step 4: loss = 13.8843 (1.015 sec/step)
INFO:tensorflow:global step 5: loss = 9.9048 (1.024 sec/step)
INFO:tensorflow:global step 6: loss = 9.4803 (0.940 sec/step)
INFO:tensorflow:global step 7: loss = 9.3944 (0.918 sec/step)
INFO:tensorflow:global step 8: loss = 8.1105 (0.947 sec/step)
INFO:tensorflow:global step 9: loss = 8.0235 (0.939 sec/step)
INFO:tensorflow:global step 10: loss = 16.0693 (0.972 sec/step)
INFO:tensorflow:global step 11: loss = 12.1528 (0.920 sec/step)
INFO:tensorflow:global step 12: loss = 11.2044 (0.883 sec/step)
INFO:tensorflow:global step 13: loss = 8.3027 (0.958 sec/step)
INFO:tensorflow:global step 14: loss = 6.9101 (0.971 sec/step)
INFO:tensorflow:global step 15: loss = 9.5765 (0.905 sec/step)
INFO:tensorflow:global step 16: loss = 9.2846 (0.955 sec/step)
INFO:tensorflow:global step 17: loss = 8.5414 (0.918 sec/step)
INFO:tensorflow:global step 18: loss = 7.9638 (0.986 sec/step)
INFO:tensorflow:global step 19: loss = 8.1441 (0.838 sec/step)
INFO:tensorflow:global step 20: loss = 7.1030 (0.836 sec/step)
INFO:tensorflow:global step 21: loss = 6.4069 (0.827 sec/step)
INFO:tensorflow:global step 22: loss = 7.7179 (0.834 sec/step)
INFO:tensorflow:global step 23: loss = 7.4246 (0.854 sec/step)
INFO:tensorflow:global step 24: loss = 7.2192 (0.834 sec/step)
INFO:tensorflow:global step 25: loss = 8.7324 (0.865 sec/step)
INFO:tensorflow:global step 26: loss = 9.2637 (0.897 sec/step)
INFO:tensorflow:global step 27: loss = 8.0228 (0.818 sec/step)
INFO:tensorflow:global step 28: loss = 7.3397 (0.861 sec/step)
INFO:tensorflow:global step 29: loss = 6.7069 (0.856 sec/step)
INFO:tensorflow:global step 30: loss = 6.4322 (0.898 sec/step)
INFO:tensorflow:global step 31: loss = 6.3612 (0.866 sec/step)
INFO:tensorflow:global step 32: loss = 6.1956 (0.857 sec/step)
INFO:tensorflow:global step 33: loss = 6.6166 (0.856 sec/step)
```


im2txt训练和评估

由于训练该模型耗时较长，所以本文对于模型的训练过程采用了github上共享出来的5M steps的预训练模型，在此基础上fine-tune。

如下图所示，在step=5.041M这点上，我们进行了一个调整：取消CNN网络权重的固定，使其开始和RNN一起被训练更新，所以我们可以看到loss曲线从一个点陡然上升，然后开始下降，Perplexity(困惑度)也是反应了网络性能的改变，并且在此之前，loss曲线的平均值在1.5浮动，而开始fine-tune后，loss逐渐升高，在网络训练了一段时间后才开始下降，并且有一点也出现很大变化，此时网络训练1step耗时从0.8 second 开始上升，逐渐变化至6 sec/step，在经过很长时间训练后才下降。



im2txt之推断

其中im2txt的推断实现(im2txt/run_inference.py)流程如下:

从checkpoint路径restore保存的模型

=> 从输入图片提取"lstm/initial_state:0"的state值 & 加载词汇表

=> 输入initial_state到LSTM, 从生成第一个字符开始, 获取到"softmax:0"的值 和 "lstm/state:0"的值, 在限制字幕maxlength下, 执行循环操作: 以LSTM上一步产生的output 和 state作为输入; 使用 beam search(size=3)对每一个输出的向量进行范围搜索, 根据概率大小排序选取TOP-3的值产生最佳的字符的输出, 直至生成完整语句向量

=> 根据词汇表, 把上一步输出的向量转为可读语句

以上为im2txt流程中的推断实现流程; 我们的模型作为API服务的实现逻辑也和推断的逻辑一样, 所以在我们的系统API服务中就采取了以上流程。

im2txt实验对比

左图为模型训练2M steps Perplexity \approx 8.9

右图为模型训练5M steps Perplexity \approx 8.4

右图对比左图会有较好的识别效果, 但整体还是有待改进

