

复旦大学

机器学习课程论文源代码 neuraltalk

阅读和实验报告

“Deep Visual-Semantic Alignments for Generating Image Descriptions”

版本	更新时间	作者	更新内容
1.0	2015/12/08	严健康	初始版本
1.1	2016/1/4	严健康	完成全部章节

严健康 14210720109

2015-12-16

目录

1. 论文和源代码简介	3
2. 源代码分析	3
2.1 源代码文件结构	3
2.2 测试集数据分析	4
2.2.1 测试集数据格式分析	4
2.2.2 测试集数据的产生	5
2.3 基本框架和参数符号约定	12
2.4 基本数据结构和类的分析	14
2.4.1 基本数据结构 model、misc 和 cache	14
2.4.2 BasicDataProvider 类	14
2.4.3 GenericBatchGenerator 类	14
2.4.4 RNNGenerator	17
2.4.5 LSTMGenerator	23
2.4.6 Solver	23
2.5 模型训练流程分析	29
2.5.1 driver.py—__main__	29
2.5.2 driver.py—main()函数	32
2.6 模型预测过程分析	37
2.6.1 测试集图片预测	38
2.6.2 自定义图片预测	39
3. 源代码运行实验	39
3.1 源代码的运行环境和数据准备	39
3.1.1 运行环境	39
3.1.2 数据准备	40
3.2 模型训练	40
3.2.1 启动训练	40
3.2.2 网页显示训练过程	41
3.3 模型预测	42
3.3.1 启动预测	42
3.3.2 网页显示预测结果	44
3.4 部分预测结果	45
4. 感想和总结	48

1. 论文和源代码简介

本次课程报告我选择的是论文是作者 Karpathy 发表在 2015 年的 CVPR 会议上的论文 *Deep Visual-Semantic Alignments for Generating Image Descriptions*^[1]。论文的主要工作内容为：

- 1) 对其在 NIPS 2014 会议上发表的论文 *Deep Fragment Embeddings for Bidirectional Image-Sentence Mapping* 中的算法进行了改进。NIPS 2014 会议上的这篇论文主要介绍了一种将图像区域与图像描述语句中的词组关系进行双向映射和搜索的算法。通过将图片和语句描述输入作者实现的神经网络模型后，图片区域与文本描述短语的对应关系。而 CVPR 2015 会议上的这篇论文对其中语句矢量模型的生成方式进行了改进，使用双向的递归神经网络（BRNN, Bidirectional recurrent neural network）来取代原有的 Dependency Tree Relations 方法，并对图像区域-描述短语之间的匹配得分函数和误差函数进行了简化，从而得到了更好的测试结果。
- 2) 实现了图像区域（或者完整图像）的语句描述短语（或者整个语句）的生成。即给定一副图像或者图像的一个区域，能够生成该图像的完整语句描述或者图像区域的短语描述。如下图所示。作者采用的模型架构为 RCNN+BRNN，其中 RCNN，用于将图片或者图片区域转化为 h 维空间向量，同时使用 word2vector 将图片对应语句中的每一个单词转化为 h 维的空间矢量，然后两者输入 RNN(Recurrent neural network)，这个递归神经网络会输出语句序列。

作者在 github 上公开了他 2015 年的论文的 Demo 源代码^[2]，源代码主要实现的功能为训练一个可以对输入图像产生完整语句描述的神经网络模型。源代码主要使用 python 语言编写。因此作者并没有公布其工作内容 1（关于图像区域和文本短语双向映射和检索的模型）的代码。而且对于工作内容 2 的测试数据也仅仅限于完整图像，而并没有考虑图像的区域。

不过，作者给出了他 2014 年论文工作的源代码，是 matlab 的代码。

由于时间精力有限，我的本次报告主要研究作者 2015 年的关于对完整图片生成文本描述的神经网络模型代码 neuraltalk（后面简称 neuraltalk 代码）。

2. 源代码分析

2.1 源代码文件结构

顶层文件结构如下表所示。

表格 1 neuraltalk 项目顶层文件列表

文件/文件夹 (无扩展名为文件夹)	功能描述
cv	用于存储模型在训练过程中产生的检查点数据
data	用于存放测试图片集的数据(具体格式后面详解)
eval	存在测试生成的语句描述结果以及计算 BLEU 分数的 perl 脚本
example_images	存放作者用于模型演示的图片和模型的生成结果
imagernn	整个神经网络模型的主要实现部分代码
matlab_features_reference	使用 caffe 产生图像的 CNN 编码矢量的 matlab 接口代码
python_features	使用 caffe 产生图像的 CNN 编码矢量的 python 接口代码
status	存放在运行优化过程中的状态暂存结果(json 格式)
vis_resources	网页显示模型训练和预测结果的资源文件
driver.py	模型训练部分的代码
eval_sentence_predictions.py	执行在测试数据 test 集上的所有图片测试的脚本
predict_on_images.py	执行在自定义图片上测试的脚本

py_caffe_feat_extract.py	使用 python 来调用 CNN 生成图片矢量的脚本文件
monitorcv.html	训练过程中的网页显示监控部分
visualize_result_struct.html	测试结果的网页直观显示
Readme.md	代码的 readme 文件
requirements.txt	代码正常运行的要求文件

其中 imagernn 文件夹中的文件列表如下：

表格 2 neuraltalk 项目 imagernn 文件夹中文件列表

文件名	功能描述
data_provider.py	读取图像和语句数据文件，提供图像语句对生成接口
utils.py	工具类函数，包括随机初始化矩阵，合并 dict 等
solver.py	整个模型的抽象层，将一批次的训练抽象为 step 函数
generic_batch_generator.py	语句描述生成模型的代理类，其中会根据参数调用特定的生成模型(LSTM 或 RNN)
lstn_generator.py	采用 LSTM 模型的语句描述生成模型
rnn_generator.py	采用 RNN 模型的语句描述生成模型
imagernn_utils.py	模型中使用到的两个辅助函数，选择 Generator 函数和验证集运行函数 eval_plt;

2.2 测试集数据分析

2.2.1 测试集数据格式分析

论文中提到的图片测试集一共有三个，分别是 flickr8，flickr30 和 MSCOCO。然后坐着通过众包的方式为每张图片产生了五句话描述。最终对于每一个测试集，测试数据都由三部分组成：

- 1) 实际的图片文件(在训练和测试中并不需要，只是在网页显示测试结果的时候需要)
- 2) 所有图片对应的向量组成的 matlab 格式文件 vgg_feats.mat。例如 flickr8 测试集的 vgg_feats.mat 中存储的是一个 4096x8000 的矩阵，因为 flickr8 中一共有 8000 张图片，每张图片对应一个 4096 维的向量；
- 3) 图片与语句的对应关系文件 dataset.json，该 json 文件的格式如下所示。

```
{
  "images":
  [
    {
      "sentids": [0,1,2,3,4],
      "imgid": 0,
      "sentences": [
        {
          "tokens": ["a","black","dog","is","running","after","a","white","dog","in","the","snow"],
          "raw": "A black dog is running after a white dog in the snow .",
          "imgid": 0,
          "sentid": 0},
        {
          "tokens": ["black","dog","chasing","brown","dog","through","snow"],
          "raw": "Black dog chasing brown dog through snow",
          "imgid": 0,
          "sentid": 1},
        {
          "tokens": ["two","dogs","chase","each","other","across","the","snowy","ground"],
          "raw": "Two dogs chase each other across the snowy ground .",
          "imgid": 0,
          "sentid": 2},
      ]
    }
  ]
}
```

```

{"tokens": ["two", "dogs", "play", "together", "in", "the", "snow"],
 "raw": "Two dogs play together in the snow .",
 "imgid": 0,
 "sentid": 3},
{"tokens": ["two", "dogs", "running", "through", "a", "low", "lying", "body", "of", "water"],
 "raw": "Two dogs running through a low lying body of water .",
 "imgid": 0,
 "sentid": 4}],
"split": "train",
"filename": "2513260012_03d33305cf.jpg" }, {.....}
..... {.....}],
"dataset": "flickr8k")

```

可以看到最顶层键是 `images` 和 `dataset`, `images` 中包含很多 `image` 对象, 每个 `image` 由 `sentids`, `imgid`, `sentences`, `split`, `filename` 五个键组成, 其中 `sentids` 是一张图片对应的五个语句的编号, `imgid` 是这张图片的编号, `sentences` 是这样图片对应的五个句子, `filename` 是这张图片在数据集中的名称; `split` 表示这张图片从属的集合, 可以是训练集 `train`、验证集 `val`、和测试集 `test`。每条语句包括 `tokens`, `raw`, `imgid`, `sentid`。其中 `tokens` 是语句的单词分割结果 (不包含符号), `raw` 是原生语句, `imgid` 是对应的图片 `id`, `sentid` 是语句 `id`。语句 `id` 和图片 `id` 都是不重复的。

2.2.2 测试集数据的产生

首先测试集图片可以直接从 Internet 上下载, 作者也提供了下载链接²。然后 `dataset.json` 文件是作者通过众包生成的, 可以从作者网站上下载。存储图片矢量的 `mat` 格式文件作者也提供了下载, 但是我们必须更加清楚这个文件是怎么生成的。

论文阅读报告中提到作者是使用 `Caffe` 工具来生成的, 并且作者也提供了生成这个 `mat` 文件的基于 `python` 和 `matlab` 封装好的脚本。下面主要介绍一下作者怎么利用 `Caffe` 来生成。

作者使用的 `CNN` 模型是牛津大学的 **Visual Geometry Group** 团队在 **ILSVRC-2014**(ImageNet Large Scale Visual Recognition Competition, ImageNet 是一个图像识别的数据库)上使用的模型^[3]。该模型在 `caffe` 的模型库中是存在的, 可以从 `caffe zoo`^[4]中下载。模型包括两部分:

1) 模型层级定义文件, 内容如下所示, 主要是定义神经网络的每一层的结构, 一共 16 层:

```

name: "VGG_ILSVRC_16_layers"
input: "data"
input_dim: 10
input_dim: 3
input_dim: 224
input_dim: 224
layers {
  bottom: "data"
  top: "conv1_1"
  name: "conv1_1"
  type: CONVOLUTION
  convolution_param {
    num_output: 64
    pad: 1
    kernel_size: 3
  }
}

```

```

}
layers {
  bottom: "conv1_1"
  top: "conv1_1"
  name: "relu1_1"
  type: RELU
}
layers {
  bottom: "conv1_1"
  top: "conv1_2"
  name: "conv1_2"
  type: CONVOLUTION
  convolution_param {
    num_output: 64
    pad: 1
    kernel_size: 3
  }
}
layers {
  bottom: "conv1_2"
  top: "conv1_2"
  name: "relu1_2"
  type: RELU
}
layers {
  bottom: "conv1_2"
  top: "pool1"
  name: "pool1"
  type: POOLING
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}
layers {
  bottom: "pool1"
  top: "conv2_1"
  name: "conv2_1"
  type: CONVOLUTION
  convolution_param {
    num_output: 128
    pad: 1
    kernel_size: 3
  }
}
layers {

```

```

    bottom: "conv2_1"
    top: "conv2_1"
    name: "relu2_1"
    type: RELU
}
layers {
    bottom: "conv2_1"
    top: "conv2_2"
    name: "conv2_2"
    type: CONVOLUTION
    convolution_param {
        num_output: 128
        pad: 1
        kernel_size: 3
    }
}
layers {
    bottom: "conv2_2"
    top: "conv2_2"
    name: "relu2_2"
    type: RELU
}
layers {
    bottom: "conv2_2"
    top: "pool2"
    name: "pool2"
    type: POOLING
    pooling_param {
        pool: MAX
        kernel_size: 2
        stride: 2
    }
}
layers {
    bottom: "pool2"
    top: "conv3_1"
    name: "conv3_1"
    type: CONVOLUTION
    convolution_param {
        num_output: 256
        pad: 1
        kernel_size: 3
    }
}
layers {
    bottom: "conv3_1"
    top: "conv3_1"

```

```

    name: "relu3_1"
    type: RELU
}
layers {
  bottom: "conv3_1"
  top: "conv3_2"
  name: "conv3_2"
  type: CONVOLUTION
  convolution_param {
    num_output: 256
    pad: 1
    kernel_size: 3
  }
}
layers {
  bottom: "conv3_2"
  top: "conv3_2"
  name: "relu3_2"
  type: RELU
}
layers {
  bottom: "conv3_2"
  top: "conv3_3"
  name: "conv3_3"
  type: CONVOLUTION
  convolution_param {
    num_output: 256
    pad: 1
    kernel_size: 3
  }
}
layers {
  bottom: "conv3_3"
  top: "conv3_3"
  name: "relu3_3"
  type: RELU
}
layers {
  bottom: "conv3_3"
  top: "pool3"
  name: "pool3"
  type: POOLING
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}

```



```

}
layers {
  bottom: "pool3"
  top: "conv4_1"
  name: "conv4_1"
  type: CONVOLUTION
  convolution_param {
    num_output: 512
    pad: 1
    kernel_size: 3
  }
}
layers {
  bottom: "conv4_1"
  top: "conv4_1"
  name: "relu4_1"
  type: RELU
}
layers {
  bottom: "conv4_1"
  top: "conv4_2"
  name: "conv4_2"
  type: CONVOLUTION
  convolution_param {
    num_output: 512
    pad: 1
    kernel_size: 3
  }
}
layers {
  bottom: "conv4_2"
  top: "conv4_2"
  name: "relu4_2"
  type: RELU
}
layers {
  bottom: "conv4_2"
  top: "conv4_3"
  name: "conv4_3"
  type: CONVOLUTION
  convolution_param {
    num_output: 512
    pad: 1
    kernel_size: 3
  }
}
layers {

```

```

    bottom: "conv4_3"
    top: "conv4_3"
    name: "relu4_3"
    type: RELU
}
layers {
    bottom: "conv4_3"
    top: "pool4"
    name: "pool4"
    type: POOLING
    pooling_param {
        pool: MAX
        kernel_size: 2
        stride: 2
    }
}
layers {
    bottom: "pool4"
    top: "conv5_1"
    name: "conv5_1"
    type: CONVOLUTION
    convolution_param {
        num_output: 512
        pad: 1
        kernel_size: 3
    }
}
layers {
    bottom: "conv5_1"
    top: "conv5_1"
    name: "relu5_1"
    type: RELU
}
layers {
    bottom: "conv5_1"
    top: "conv5_2"
    name: "conv5_2"
    type: CONVOLUTION
    convolution_param {
        num_output: 512
        pad: 1
        kernel_size: 3
    }
}
layers {
    bottom: "conv5_2"
    top: "conv5_2"

```

```

    name: "relu5_2"
    type: RELU
}
layers {
  bottom: "conv5_2"
  top: "conv5_3"
  name: "conv5_3"
  type: CONVOLUTION
  convolution_param {
    num_output: 512
    pad: 1
    kernel_size: 3
  }
}
layers {
  bottom: "conv5_3"
  top: "conv5_3"
  name: "relu5_3"
  type: RELU
}
layers {
  bottom: "conv5_3"
  top: "pool5"
  name: "pool5"
  type: POOLING
  pooling_param {
    pool: MAX
    kernel_size: 2
    stride: 2
  }
}
layers {
  bottom: "pool5"
  top: "fc6"
  name: "fc6"
  type: INNER_PRODUCT
  inner_product_param {
    num_output: 4096
  }
}
layers {
  bottom: "fc6"
  top: "fc6"
  name: "relu6"
  type: RELU
}
layers {

```

```

bottom: "fc6"
top: "fc6"
name: "drop6"
type: DROPOUT
dropout_param {
  dropout_ratio: 0.5
}
}
layers {
  bottom: "fc6"
  top: "fc7"
  name: "fc7"
  type: INNER_PRODUCT
  inner_product_param {
    num_output: 4096
  }
}
layers {
  bottom: "fc7"
  top: "fc7"
  name: "relu7"
  type: RELU
}
}

```

2) 参数权重文件。可以从其官方网站下载 http://www.robots.ox.ac.uk/~vgg/research/very_deep/ (528M for 16-layer)。

有了这两个文件后，可以构建一个 CNN 网络，作者使用 matlab 代码来编写的这个模块，主要代码位于：
matlab_features_reference/extract_features.m。

首先初始化一个CNN网络：matcaffe_init(use_gpu, model_def_file, model_file);
然后读取要处理的一批次图片，即读取 RGB 三维数组[R,G,B]，如果是灰度图片，转为三维的[I,I,I]。
然后调用 caffe 的 forward 函数，处理一批次的图片输入，返回计算结果。
所有计算结果组成一个(4096 x image_nums)的矩阵，然后把结果写入到 vgg_feats.mat 文件中即可。

2.3 基本框架和参数符号约定

前面说到 neuraltalk 代码主要实现了对完整图像生成语句描述的神经网络模型，neuraltalk 代码中给出了两种模型的实现，分别是 RNN 和 LSTM（因为谷歌公司的论文^[5]中采用的 LSTM 模型表现更加出色，所以作者马上跟进并且实现了 LSTM 架构模型）。我们先只考虑 RNN 生成模型。那么基本架构如下图所示。

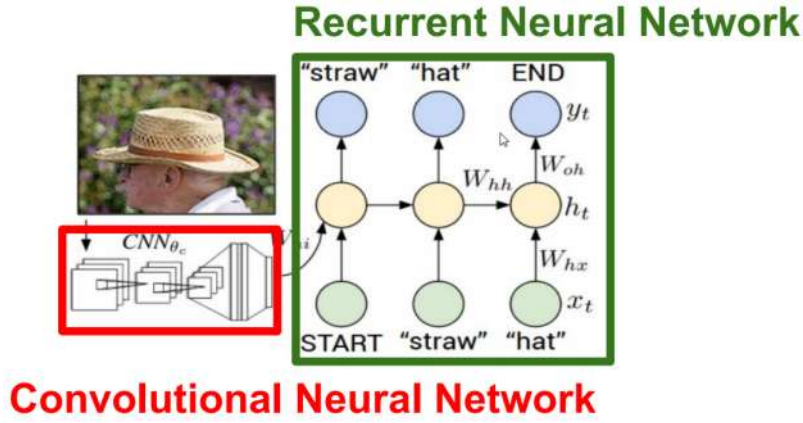


图 2-1 生成图像描述语句的算法框图

- 1) CNN 网络采用 Caffe 工具生成得到。对于每张图片将会生成一个 4096 维的向量。首先将这个向量映射到一个 image_encoding_size（默认值为 256）维的向量，采用以下转换过程：

$$X_i = [CNN_{\theta_c}(\text{Image})] \cdot W_e + b_e$$

其中 $CNN_{\theta_c}(\text{Image})$ 是 1×4096 的向量， W_e 是 4096×256 的矩阵， b_e 是 1×256 的矩阵。同时记 X_e 为一批次的图片所有图片矢量按行拼接成的矩阵 $[\text{batch_size} \times 4096]$ 。

- 2) 然后将每个单词转化为 word_encoding_size（默认为 256）维的向量，flickr8 的所有图片对应的语句单词库中出现次数大于等于 5 的一共有 2538 个单词，采用以下转换过程：

$$\begin{aligned} W_s &= \text{random_array}(\text{vocabulary_size} \times \text{word_encoding_size}) \\ &= \text{random_array}(2538 \times 256) \end{aligned}$$

$$X_{si} (\text{index}=i) = W_s[i]$$

即单词的编码采用随机编码。首先随机初始化一个 2538×256 的矩阵，然后每个单词对应一个索引 i ，直接取 W_s 的第 i 行即为该单词的编码。因此对于一个含有 n 个单词(包含 #START#)的语句，可以编码为 $n \times 256$ 维的向量 X_s 。

- 3) 然后是输入图像矢量和语句矢量到 RNN 网络中，在论文阅读报告 3.2.2 节中给出了 RNN 网络的计算过程，这里再次计算如下：

$$X_{sh} = X_s W_{xh} + b_{xh}$$

如果输入经过 ReLU 的话，

$$X_{sh} = \text{ReLU}(X_{sh})$$

$$X_i = \text{ReLU}(X_i)$$

$$H_t = \text{ReLU}(X_{sh} + H_{t-1} \cdot W_{hh} + b_{hh} + \Gamma(t=0) \cdot X_i), \quad t = 0, 1, \dots, n-1$$

$$\Gamma(t=0) = \begin{cases} 1, & t=0 \\ 0, & \text{other} \end{cases}$$

$$H_{-1} = [0] \quad (n \times 256)$$

$$Y = H_t \cdot W_d + b_d$$

$$P = \text{softmax}(Y)$$

其中 X_{sh} 是 $n \times 256$ 的矩阵 (n 为语句单词数目)， W_{xh} 是 256×256 的矩阵， b_{xh} 是 1×256 的向量， H_t 是 $n \times 256$ 的矩阵， W_{hh} 是 256×256 的矩阵， b_{hh} 是 1×256 的向量。 W_d 是 256×2538 的矩阵， b_d 是 1×2538 维的向量， Y 和 P 都是 $n \times 2538$ 维的矩阵。

这样就得到了最终的输出，然后就是计算误差并反向传播误差求梯度的过程。

这样需要注意两点：

- 1) 作者在论文中和在代码中使用的参数符号不相同，但基本原理是相同的，要注意区别开；最终使用 RNN 模型的语句生成器需要训练的参数为： W_e ， b_e ， W_s ， W_{xh} ， b_{xh} ， W_{hh} ， b_{hh} ， W_d ， b_d 。
- 2) 作者在论文中提到其单词向量的产生是使用 word2vec 工具^[6]编码而来。但在实际的代码中使用的是随机编

码。作者也在论文中提到过，在实际的测试中发现使用随机编码会有更好的性能表现。

最后，为了便于后文的展开和简洁，后文中的参数符号全部沿用本章节的定义。

2.4 基本数据结构和类的分析

2.4.1 基本数据结构 model、misc 和 cache

在整个模型的代码中，会经常见到 model、misc 和 cache 这几个变量，他们其实都是字典类型的结构。

model 中用于存放语句生成模型中所有需要训练的参数，比如 RNN 模型中，model 则存放 W_e , b_e , W_s , W_{xh} , b_{xh} , W_{hh} , b_{hh} , W_d , b_d 等参数名称及其对应的值。model 会作为函数参数在很多类之间进行传递。

misc 字典结构也是用于存放需要在框架间传递的数据信息，其中主要包括：indexToWord 和 wordToIndex，即单词库中单词与索引的双向映射；‘update’，update 这个键值对应的是在模型训练过程中需要更新的超参数名，如 RNN 网络中需要更新 W_e , b_e , W_s , W_{xh} , b_{xh} , W_{hh} , b_{hh} , W_d , b_d ；‘regularize’，regularize 这个键值对应的是在模型训练过程中需要进行 L2 范式惩罚的参数名，如 RNN 网络中需要惩罚 W_e , W_s , W_{xh} , W_{hh} , W_d ；

cache 数据结构则主要是用作缓存，缓存主要体现在两方面：1) 因为我们采用的批量的随机梯度下降，并且是每批次后进行一次反向传播的参数更新，因此需要把一批次中每个图像语句对运行时的中间数据缓存下来，便于后期一批次的反向传播的梯度计算；2) 由于作者使用了代理的 GenericBatchGenerator，因此计算必然会在底层和代理层之间传递，因此需要使用缓存来缓存数据。缓存的内容主要是训练过程中的参数和输入输出，以及一些优化参数等，这些将在代码分析时提到。

2.4.2 BasicDataProvider 类

BasicDataProvider 类的主要功能是读取测试集的图片 and 文本数据，并且给网络模型提供数据接口。BasicDataProvider 包含以下 10 个方法：

- 1) **__init__(dataset)**: 构造函数，读取 dataset.json 到 self.dataset 中，读取 vgg_feats.mat 到 self.features 中，并将 self.dataset 中的 images 中的 img 按 split 类型(train, val, test)分类存放；
- 2) **_getImage(img)**: 读取 img 对应的向量；
- 3) **_getSentences(sent)**: 获取 sent 对应的语句；
- 4) **getSplitSize(split, ofwhat = ‘sentences’)**: 获取 split 集中 ofwhat 的数目(默认是语句，否则是图片)；
- 5) **sampleImageSentencePair(split = ‘train’)**: 获取 split 集(默认 train)中随机的图像-语句对；
- 6) **iterImageSentencePair(split = ‘train’, max_images = -1)**: 获取 split 集(默认 train)中不多于 max_images 的图像语句对，返回迭代器结构；
- 7) **iterImageSentencePairBatch(split = ‘train’, max_images = -1, max_batch_size = 100)**: 批量(默认 100 对)获取 split 集(默认 train)中的图像语句对，但不得多于 max_images，返回迭代器结构；
- 8) **iterSentences(split = ‘train’)**: 产生所有的语句迭代器
- 9) **iterImages(split = ‘train’, shuffle = False, max_images = -1)**: 产生所有图片的迭代器，shuffle 选项表示乱序；
- 10) **getDataProvider(dataset)**: 如果 dataset 类型支持的话，返回 BasicDataProvider 类的实例。

2.4.3 GenericBatchGenerator 类

GenericBatchGenerator 类是一个代理类，主要包括四个方法(init(), forward(), backward(), predict())。它是对 RNN 和 LSTM 两种模型的不同参数、相同接口特性的统一封装。因为 RNN 和 LSTM 的不同主要在于网络结构和训练参数的不同，但是输入输出和训练过程是相同的，都是首先将图片向量和语句向量编码，然后输入模型进行正向的结果计算，最后进行反向的误差传播和参数更新。因此 GenericBatchGenerator 提取了两者相同的正向和反向接口和输入输出的编解码，使得代码结构更加易于扩展，但实际中还是调用底层的 RNN 或者 LSTM 的相应函数。GenericBatchGenerator 中存在一个 decodeGenerator 函数，将根据运行参数中的 generator 来选取对应的网络模型。

GenericBatchGenerator 模型的主要方法如下：

- 1) **init(params, misc)**: 传入参数为用户运行参数和 misc 数据(主要存在单词库的单词索引信息等框架间传递数

据); 函数主要功能为初始化模型参数: 随机初始化 W_e 和 W_s , 零初始化 b_e ; 然后通过命令行参数中的 generator 的内容来 decodeGenerator 确定实际的 Generator 到底是 RNNGenerator 还是 LSTMGenerator。并且调用其实际的 init 函数; 并将两者的 model 和 misc 结构合并;

2) **forward(batch, model, params, misc, predict_mode = False):** 传入参数为一批次的图像语句对 batch, 模型参数 model, 用户运行参数 params, 框架间传递数据 misc, 当前是否预测模式的标志;

函数的主要功能为执行一批次训练数据的计算, 其运行流程如下图所示。其中, 因为每一张图片都是一个 4096 维的向量 X_i , 因此可以将一批次的图片组成一个矩阵 F, 利用 numpy 矩阵乘法一次计算出所有的图像编码 X_e (batch_size x image_encoding_size); 语句编码即为将语句中每个单词的编码按行组成矩阵; 调用底层 generator 即为调用 RNNGenerator 或者 LSTMGenerator; 最终保存的 cache 缓存数据主要内容为: 一批次数据计算过程中的所有缓存 gen_caches (视模型而异), W_e , $W_s.shape$, F, generator_str(用于确定底层生成器模型); 当前是否预测模式的标志是用于指示当前是否是训练过程, 如果是训练过程, 那么需要保存缓存数据用于反向传播, 如果是预测模式, 那么不存在反向传播过程, 不需要保存缓存数据。返回的计算 Ys 为一个数组, 数组长度为 batch_size, 数组中的每一项都是 ni x 2538 的矩阵(ni 代表 batch 中的第 i 个语句对应的单词数目)。

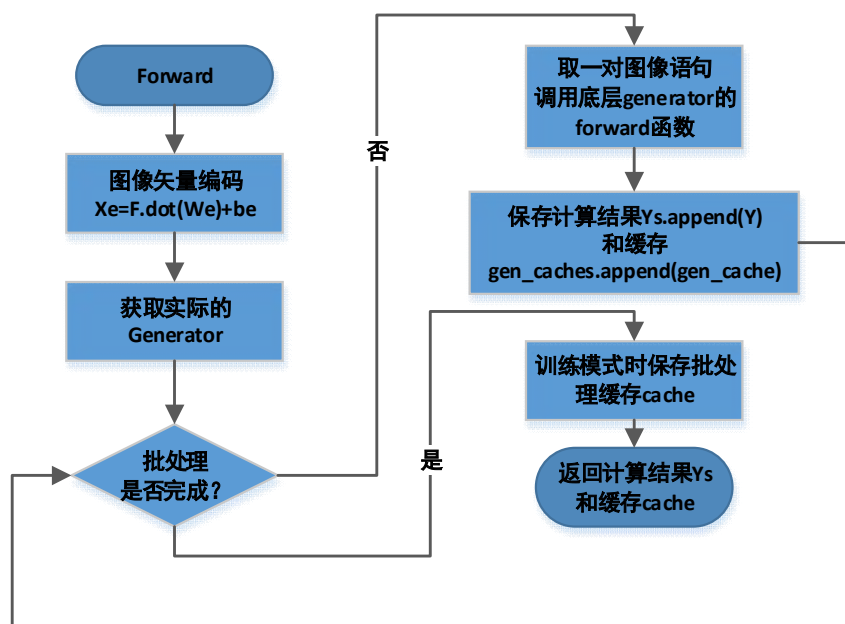


图 2-2 GenericBatchGenerator 的 forward 函数流程图

3) **backward(dY, cache):** 传入参数为计算出的批量输出误差 dY(batch_size x)和前向过程 forward 的缓存数据 cache; 函数的功能主要是计算误差对各参数的梯度。函数的运行流程如下图所示。

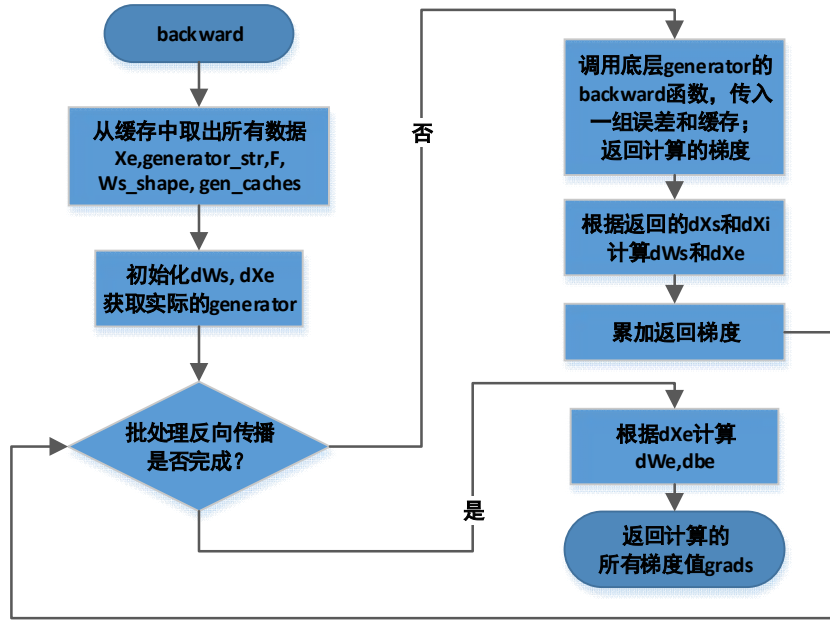


图 2-3 GenericBatchGenerator 的 backward 函数流程图

因为反向误差是从输出到输入逐级传播的，同时是批量的随机梯度下降算法，因此进入一个 for 循环，循环次数为 `batch_size`，首先将输出误差 `dY[i]` 和相应缓存 `gen_cache` 传入底层的 Generator，等待底层的 `backward` 计算完毕后会返回计算的所有参数的梯度值，如果是 RNN 的话，那么这些梯度值为 `dWxh`，`dbxh`，`dWhh`，`dbhh`，`dWd`，`dbd`，`dXs`，`dXi`。其中 `GenericBatchGenerator` 会将其中的 `dXe`，`dXi` 取出计算 `dWs` 和 `dXe`，并将所有结果累加后存储(每一次都会计算出这些参数的梯度，批量梯度下降即累加起来即可)，一直到批量循环结束。

因为 $X_i = X_e[i, :]$ ， $i = 0, 1, \dots, \text{batch_size} - 1$ ，所以 $dX_e[i, :] = dX_i$ ；

因为 $X_s[n \times \text{word_encoding_size}] = W_s[\text{vocab_size} \times \text{word_encoding_size}]$ 中按语句单词序取 n 个向量拼接，因此 $dW_s[j, :] = dX_s[k]$ ，语句的第 k 个单词的索引为 j 。

循环结束后，因为：

$$X_e = F \cdot W_e + b_e$$

$$F = \begin{bmatrix} CNN_{\theta_c}(\text{Image}_0) \\ \dots \\ CNN_{\theta_c}(\text{Image}_{\text{batch_size} - 1}) \end{bmatrix}$$

所以根据反向传播规则有：

$$dW_e = F^T \cdot dX_e$$

$$db_e = \begin{bmatrix} \sum_{i=0}^{\text{batch_size}-1} dX_e[i, 0] & \dots & \sum_{i=0}^{\text{batch_size}-1} dX_e[i, 2537] \end{bmatrix}$$

最后 `backward` 函数返回 `dWe`，`dbe`，`dWs` 和底层 Generator 模型计算处的参数梯度。

4) **predict(batch, model, params, **kwparams)**: 传入参数为传入参数为一批次的图像数据 `batch`，模型参数 `model`，用户运行参数 `params`，以及其他参数。

函数的主要功能为：基本与前向传播计算 `forward` 相似。

首先是图像矢量的编解码：

$$X_e = F \cdot W_e + b_e$$

其中输入图片数据 F 为 `batch_size` x 4096 的矩阵，得到的 X_e 为 `batch_size` x `image_encoding_size` 的矩阵。

然后对 X_e 中的每一行，调用底层 Generator 的 `predict` 计算得到结果 `gen_Y`，最后所有的 `gen_Y` 组成 Y_s 返回。

注意：在 `predict` 函数中没有缓存 `cache`，因为在 `predict` 模式时并不需要缓存数据来做反向传播过程。

返回的 Y_s 中的每一项 gen_Y 都是预测出来的一条或多条语句，其结构为：[[预测误差，语句对应单词索引数组]]。具体可参看 RNNGenerator 的 predict 函数。

2.4.4 RNNGenerator

RNNGenerator 是对 RNN 模型的基本实现。与其代理类 GenericBatchGenerator 的方法类似，其主要功能函数分为以下四个：

- 1) **init(input_size, hidden_size, output_size):** 模型参数初始化函数。输入参数分别为：input_size(输入向量维度)，hidden_size(隐层的向量维度)，output_size(输出层的向量维度)。本文参数符号约定一节已经给出了 RNN 网络的计算模型，因此需要初始化的参数为 W_{xh} , b_{xh} , W_{hh} , b_{hh} , W_d , b_d 。
- 2) **forward (Xi, Xs, model, params, **kwargs):** 前向计算函数。输入 Xi 为编码后的图像矢量，Xs 为编码后的语句矩阵，model 为模型参数，params 为运行参数，其他可选参数 kwargs(实际包括 predict_mode，即指示当前是否是预测模式还是训练模式)。

这里需要先介绍一下 **dropout 优化技巧**^[7]。训练神经网络模型时，如果训练样本较少，为了防止模型过拟合，Dropout 可以作为一种技巧供选择。Dropout 是 Hinton 最近 2 年提出的，源于其文章 Improving neural networks by preventing co-adaptation of feature detectors。Dropout 是指在模型训练时随机让网络某些隐含层节点的权重不工作，不工作的那些节点可以暂时认为不是网络结构的一部分，但是它的权重得保留下来（只是暂时不更新而已），因为下次样本输入时它可能又得工作了。由于每次用输入网络的样本进行权值更新时，隐含节点都是以一定概率随机出现，因此不能保证每 2 个隐含节点每次都同时出现，这样权值的更新不再依赖于有固定关系隐含节点的共同作用，阻止了某些特征仅仅在其它特定特征下才有效果的情况。Hinton 在他的论文中给出了一系列实验证明了 dropout 可以阻止过拟合。Dropout 在使用时实际上就是将一部分输入或者输出置为 0。

函数详细计算过程如下：

如果使用 dropout 编码的话，从参数中获取 drop_prob_encoding 比例因子，计算 dropout 掩码并应用到输入上：

$$U_i = [\mathbf{4096} \times (\text{random}(0,1) > \text{drop_prob_encoding} ? \text{scale} : 0)]$$

$$X_i = X_i \times U_i$$

$$U_s = [\mathbf{n} \times \text{word_encoding_size} \times (\text{random}(0,1) > \text{drop_prob_encoding} ? \text{scale} : 0)]$$

$$X_s = X_s \times U_s$$

输入语句向量编码：

$$X_{sh} = X_s W_{xh} + b_{xh}$$

如果输入经过 ReLU 激活函数的话：

$$X_{sh} = \text{ReLU}(X_{sh})$$

$$X_i = \text{ReLU}(X_i)$$

然后开始进行 RNN 网络的 n 步长迭代计算：

$$H_t = \text{ReLU}(X_{sh} + H_{t-1} \cdot W_{hh} + b_{hh} + [\Gamma(t=0) \parallel (\text{not rnn_feed_once}) \cdot X_i]), t = 0, 1, \dots, n-1$$

$$\Gamma(t=0) = \begin{cases} 1, & t=0 \\ 0, & \text{other} \end{cases}$$

$$H_{-1} = [0] \quad (\mathbf{n} \times 256)$$

如果输出需要 dropout，从参数中获取 drop_prob_decoding 比例因子，计算 dropout 掩码并应用到输入上

$$U_2 = [\mathbf{n} \times \text{vocab_size} \times (\text{random}(0,1) > \text{drop_prob_encoding} ? \text{scale} : 0)]$$

$$H_t = H_t \times U_2$$

最后计算输出向量

$$Y = H \cdot W_d + b_d$$

其中有一个与前面不同的地方在于 **rnn_feed_once** 变量，这个变量用于控制图像矢量是否仅在 $t=0$ 时刻输入 RNN 网络，还是在每个时刻都输入系统。

另外前面提到最终的输出将会经过 **softmax** 函数输出，但 softmax 的计算并不在 Generator 内部，而是在 solver 中调用 RNNGenCost 函数来计算的，在介绍 Solver 将会提到。

对应的代码如下所示，可以看到两者相符：

```
#取出 dropout 编解码比率参数
drop_prob_encoder = params.get('drop_prob_encoder', 0.0)
drop_prob_decoder = params.get('drop_prob_decoder', 0.0)
relu_encoders = params.get('rnn_relu_encoders', 0)
rnn_feed_once = params.get('rnn_feed_once', 0)

#执行 dropout 过程
if drop_prob_encoder > 0: # if we want dropout on the encoder
    # inverted version of dropout here. Suppose the drop_prob is 0.5, then during training
    if not predict_mode: # and we are in training mode
        scale = 1.0 / (1.0 - drop_prob_encoder)
        Us = (np.random.rand(*Xs.shape)) < (1 - drop_prob_encoder)) * scale # generate scaled
mask
        Xs *= Us # drop!
        Ui = (np.random.rand(*Xi.shape)) < (1 - drop_prob_encoder)) * scale
        Xi *= Ui # drop!

#输入语句向量编码
Wxh = model['Wxh']
bxh = model['bxh']
Xsh = Xs.dot(Wxh) + bxh

#执行 RELU
if relu_encoders:
    Xsh = np.maximum(Xsh, 0)
    Xi = np.maximum(Xi, 0)

#执行 n 步长的迭代
# recurrence iteration for the Multimodal RNN similar to one described in Karpathy et al.
d = model['Wd'].shape[0] # size of hidden layer
n = Xs.shape[0]
H = np.zeros((n, d)) # hidden layer representation
Whh = model['Whh']
bhh = model['bhh']
for t in xrange(n):
    prev = np.zeros(d) if t == 0 else H[t-1]
    if not rnn_feed_once or t == 0:
        # feed the image in if feedonce is false. And it it is true, then
        # only feed the image in if its the first iteration
        H[t] = np.maximum(Xi + Xsh[t] + prev.dot(Whh) + bhh, 0) # also ReLU
```

```

else:
    H[t] = np.maximum(Xsh[t] + prev.dot(Whh) + bhh, 0) # also ReLU

#输出的 dropout
if drop_prob_decoder > 0: # if we want dropout on the decoder
    if not predict_mode: # and we are in training mode
        scale2 = 1.0 / (1.0 - drop_prob_decoder)
        U2 = (np.random.rand(*H.shape)) < (1 - drop_prob_decoder)) * scale2
        H *= U2 # drop!

#输出解码
Wd = model['Wd']
bd = model['bd']
Y = H.dot(Wd) + bd

```

Forward 函数最后返回计算出来的 Y 和相应过程中的缓存数据，缓存数据包括 W_{xh} , W_{hh} , W_d , H, X_s , X_{sh} , X_i , drop_prob_encoder, drop_prob_decoder, relu_encoder, rnn_feed_once，因为这些数据将会在计算反向传播时用上。

3) backward(dY, cache): 反向传播计算梯度函数。传入参数为计算的输出误差 dY 和相应的 cache 缓存数据。dY 是一个 nx2538 的矩阵（n 为语句单词数目）。

函数的基本步骤如下：

首先从 cache 取出所有需要使用的参数， W_{xh} , W_{hh} , W_d , H, X_s , X_{sh} , X_i , drop_prob_encoder, drop_prob_decoder, relu_encoder, rnn_feed_once。

计算输出层的反向传播如下：

$$\begin{aligned}
 &\text{因为 } Y = H \cdot W_d + b_e, \text{ 所以} \\
 &dW_d = H^T \cdot dY \\
 &db_e = \left[\sum_{i=0}^{n-1} dY[i, 0], \quad \dots, \quad \sum_{i=0}^{n-1} dY[i, 2537] \right] \\
 &dH = dY \cdot W_d^T
 \end{aligned}$$

如果输出层经过了 dropout，那么要乘以 dropout 因子：

$$dH = dH \times \text{cache}[U_2]$$

然后计算递归神经网络的中间层传播，这里遵循的规则是我们在论文阅读报告中提到的基于时间展开的反向传播算法，这里的步长即为语句单词数目 n。循环 n 次，即 t 从 n-1, n-2, 到 0，依次执行以下步骤：

如果 forward 时输出经过了 RELU，那么

$$dh_t = (H[t] > 0) \times dH[t]$$

如果 t==0 或者 rnn_feed_once 为 False，那么

$$dX_i = dX_i + dh_t$$

$$dX_{sh}[t] = dh_t$$

$$db_{hh}[0] = db_{hh}[0] + dh_t$$

如果 t>0，那么：

$$\begin{aligned}
 dh_{t-1} &= dh_{t-1} + dH[t] \cdot W_{hh}^T \\
 dW_{hh} &= dW_{hh} + H[t-1] \times dh_t
 \end{aligned}$$

迭代循环完成后，如果输入经过了 RELU 激活，那么：

$$dX_{sh} = (X_{sh} > 0) \times dX_{sh}$$

$$dX_i = (X_i > 0) \times dX_i$$

注意：以上的 $(Array > 0)$ 将会返回一个与向量 $Array$ 相同维数的向量，其中大于 0 的数变为 1，小于 0 的数成为 0。

然后对语句编码进行反向传播：

$$dW_{xh} = X_s^T \cdot dX_{sh}$$

$$db_{xh} = \left[\sum_{i=0}^{n-1} dX_{sh}[i, 0], \quad \dots, \quad \sum_{i=0}^{n-1} dX_{sh}[i, 255] \right]$$

$$dX_s = dX_{sh} \cdot W_{xh}^T$$

最后，如果输入向量使用了 dropout 优化，那么：

$$dX_i = dX_i \times cache[U_i]$$

$$dX_s = dX_s \times cache[U_s]$$

这部分对应的代码如下所示。

```
# backprop the decoder
dWd = H.transpose().dot(dY)
dbd = np.sum(dY, axis=0, keepdims = True)
dH = dY.dot(Wd.transpose())

# backprop dropout, if it was applied
if drop_prob_decoder > 0:
    dH *= cache['U2']

# backprop the recurrent connections
dXsh = np.zeros(Xsh.shape)
dXi = np.zeros(d)
dWhh = np.zeros(Whh.shape)
dbhh = np.zeros((1,d))
for t in reversed(xrange(n)):
    dht = (H[t] > 0) * dH[t] # backprop ReLU

    if not rnn_feed_once or t == 0:
        dXi += dht # backprop to Xi

    dXsh[t] += dht # backprop to word encodings
    dbhh[0] += dht # backprop to bias

    if t > 0:
        dH[t-1] += dht.dot(Whh.transpose())
        dWhh += np.outer(H[t-1], dht)

if relu_encoders:
    # backprop relu
    dXsh[Xsh <= 0] = 0
    dXi[Xi <= 0] = 0
```

```

# backprop the word encoder
dWxh = Xs.transpose().dot(dXsh)
dbxh = np.sum(dXsh, axis=0, keepdims = True)
dXs = dXsh.dot(Wxh.transpose())

if drop_prob_encoder > 0: # backprop encoder dropout
    dXi *= cache['Ui']
    dXs *= cache['Us']

```

至此，RNN 模型的所有参数的误差反向传递已经完成，backward 函数计算出的误差 dW_{xh} , db_{xh} , dW_{hh} , db_{hh} , dW_d , db_d , dX_s , dX_i 打包成 dict 结构返回。

4) **predict (Xi, model, Ws, params, **kwargs)**: 预测函数的输入为编码后的图像矢量 **Xi**，模型参数 **model**，单词库初始化向量 **Ws**，运行参数 **params**，其他参数 **kwargs**。

Predict 的计算步骤为：

- 首先从参数 **params** 中取出：beam 搜索的 size，rnn_relu_encoders, rnn_feed_once，从 **model** 中取出参数： W_{xh} , W_{hh} , W_d , b_d , b_{xh} , b_{hh} ；
- 如果 **relu_encoders** 的话，对 **Xi** 应用 RELU 激活；
- **Predict** 可选择采用 **beam** 搜索算法来得到若干个条件概率较大的 **candidate sentences**。

输入 RNN 网络的第一个单词为 #START#，因此首先编码第一个单词向量：

$$X_{sh} = W_s[0] \cdot W_{xh} + b_{xh}$$

每个单词输入，RNN 网络都会计算一个对应的输出 Y_t ：

$$H_t = \text{ReLU}(X_{sh} + H_{t-1} \cdot W_{hh} + b_{hh} + [\Gamma(t=0) \parallel (\text{not rnn_feed_once})] \cdot X_i)$$

$$Y_t = H_t \cdot W_d + b_d$$

Y_t 是一个 1×2538 的向量，对 Y_t 做 softmax 处理，得到：

$$P_t = \text{softmax}(Y_t)$$

即：

$$Y_{\max} = \max_{j \in \{0, 1, \dots, 2537\}} (Y_t[j])$$

$$P_t[i] = \frac{e^{(Y_t[i] - Y_{\max})}}{\sum_{j=0}^{2537} e^{(Y_t[j] - Y_{\max})}}$$

可以求得 P_t 最大值所在的 index (因为 $Y_t[i] - Y_{\max} \leq 0$ ，所以 $P_t[i]$ 越大，误差越小)，该 index 即为计算结果 Y_t 对应的单词在单词库中的 index 索引。

对于不采用 beam search 的情况下：在 RNN 网络运行过程中，每个周期可以计算得到一个单词 index，如果单词 index=0 (代表遇到了句号输出) 或者输出单词个数超过 20，就停止 RNN 网络，这样可以得到一个单词 index 的数组 **predix** 和所有次运行计算累加的误差： $\text{predlogprob} = \sum_t (P_t[\text{index}])$ ，函数即返回 $[(\text{predix}, \text{predlogprob})]$ 。不采用 **beam search** 的方法对于每张图片只能得到一个输出语句描述，这种情况下描述内容可能是不全面的。

采用 beam search 的情况下：定义 **beams**=[(误差, 单词 index 数组, ht)] 同时定义 **beam_candidates**=[(误差, 单词 index 数组, ht)]。

首先输入第一个单词 #START#，可以计算得到 P_t 数组，这时候我们取 P_t 中最大的 **beam_size** 项，即误差最小的

beam_size 个，计算其对应的索引，并将其结果放入 beam_candidates 中，即：

`beam_candidates.append((Pt[i], [i], h1))`, i 为计算出的三个索引值

然后将 beam_candidates 中的 beam_size 项放入 beams 中，把 beam_candidates 清空；

对 beams 中的每一项，继续算出 P_t 数组，且取 beam_size 项，计算其对应的索引，并将其结果放入 beam_candidates，即：

`beam_candidates.append((Pt[j], [i, j], h2))`, j 为对计算出的三个索引值，i 是上一次计算的

这样可以得到 beam_size² 个结果，取其中误差最小的 beam_size 项，并赋给 beams，清空 beam_candidates。

这样一次类推，假设我们设定最多计算语句长度为 nsteps，以上循环 nsteps 次后，可以得到 beam_size 个长度为 nsteps (或者小于，应该如果遇到句号不会再去计算后续单词) 的结果并且存储在 beams 中。

我们将 beams 中的每一项取出，可以得到最终的 beam_size 预测语句序列及其对应的误差。

采用 beam 搜索后，可以输出多条语句，而且是从语句开头启发式查找的，有助于我们更丰富的描述图片的内容。

beam 搜索的相应代码如下所示：

```
if beam_size > 1:
    # perform beam search
    beams = [(0.0, [], np.zeros(d))]
    nsteps = 0
    while True:
        beam_candidates = []
        for b in beams:
            ixprev = b[1][-1] if b[1] else 0
            if ixprev == 0 and b[1]:
                # this beam predicted end token. Keep in the candidates but don't expand it out any
                more

                beam_candidates.append(b)
                continue
            # tick the RNN for this beam
            Xsh = Ws[ixprev].dot(Wxh) + bxh
            if relu_encoders:
                Xsh = np.maximum(Xsh, 0)
            if (not rnn_feed_once) or (not b[1]):
                h1 = np.maximum(Xi + Xsh + b[2].dot(Whh) + bhh, 0)
            else:
                h1 = np.maximum(Xsh + b[2].dot(Whh) + bhh, 0)
            y1 = h1.dot(Wd) + bd

            # compute new candidates that expand out from this beam
            y1 = y1.ravel() # make into 1D vector
            maxy1 = np.amax(y1)
            e1 = np.exp(y1 - maxy1) # for numerical stability shift into good numerical range
            p1 = e1 / np.sum(e1)
            y1 = np.log(1e-20 + p1) # and back to log domain
            top_indices = np.argsort(-y1) # we do -y because we want decreasing order
            for i in xrange(beam_size):
```

```

wordix = top_indices[i]
beam_candidates.append((b[0] + y1[wordix], b[1] + [wordix], h1))

beam_candidates.sort(reverse = True) # decreasing order
beams = beam_candidates[:beam_size] # truncate to get new beams
nsteps += 1
if nsteps >= 20: # bad things are probably happening, break out
    break
# strip the intermediates
predictions = [(b[0], b[1]) for b in beams]

```

最终返回一个数组，每一项都是<预测误差，语句对应的单词索引数组>。

2.4.5 LSTMGenerator

LSTM 的基本函数与 RNN 的相同，具体不同点在计算输出 Y 和误差偏导的方法不同，由于时间有限，暂未详细查看其代码。

2.4.6 Solver

Solver 类是对整个模型的接口封装以及梯度求解优化方法的统一接口处理。首先，我们知道为了训练这个模型，如果只有 Generator 接口的话，那么，完成一批次的训练至少需要：**参数初始化 -> 前向计算 -> 误差计算 -> 反向传播和梯度求解 -> 参数更新**。这样的接口还是稍微复杂，所以 Solver 提供了一个 step 接口，step 函数执行一个批次的数据处理和参数更新，并返回误差。

其中参数初始化 init，前向计算 forward 和反向传播求解梯度 backward 都是在 Generator 中完成，因为这些函数的基本结构在模型确定后也基本确定，而**误差计算**和**参数更新**却存在很多优化的可能，因此需要单独把误差计算和参数更新作为可扩展的接口。这也是 Solver 的目的之一。

Solver 类中的基本函数为构造函数__init__()，step()函数，梯度检查函数 gradCheck()。

构造函数很简单，只是初始化了两个缓存的字典结构 step_cache 和 step_cache2，这两个结构作为成员变量存在于整个训练过程中，并且在梯度优化的时候会用到。

2.4.6.1 误差计算函数 RNNGenCost

为了模型的可扩展性，RNN 模型的误差计算函数是在 driver.py 文件中单独定义的，并且作为参数传递给 Solver 的 step 函数。其误差函数定义为 RNNGenCost(batch, model, params, misc)。输入参数为为一批次的图像语句对 batch，模型参数 model，程序运行参数 params，框架间传递参数 misc。

RNNGenCost 函数的运行流程如下图所示。

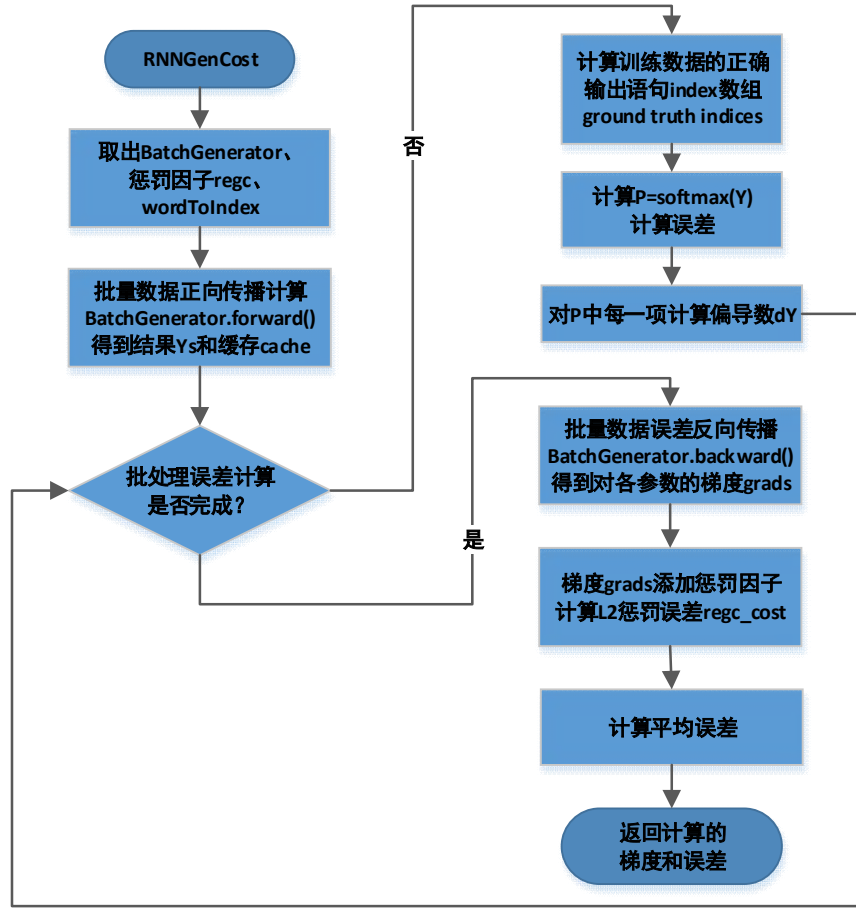


图 2-4 误差计算函数 RNNGenCost 执行流程图

其中 groud truth indices 为训练数据的基准输出语句的对应单词索引数组，记为 gtoix；

1) 计算 softmax 误差过程如下：（此处考虑一个图像语句对，批次数据计算误差和长度都应累加）

首先前向传播过程可以计算得到 $Y[n \times 2538]$ (2538 为单词库总数)，然后首先使用 softmax 函数激活。首先计算最大值：

$$Y_{\max} = \left[\max_{j \in 0,1,\dots,2537} (Y[i,j]) \right], i \in 0,1,\dots,n-1$$

得到 Y_{\max} 是 $n \times 1$ 的向量，然后

$$P[i,j] = \frac{e^{(Y[i,j]-Y_{\max}[i])}}{\sum_{j=0}^{2537} e^{(Y[i,j]-Y_{\max}[i])}}$$

其中减去 Y_{\max} 可以将输出取值限定在 $[0,1]$ 范围内，可以提高数值稳定性，而且不影响最终的误差计算。

然后计算误差，其中 $gtoix[i]$ 即为基准语句的单词索引，因为采用 softmax 之后，如果计算得到的概率 P 的第 i 行中是基准语句的单词索引这一列的概率最高，那么说明误差最小。理想情况就是 $P[i, gtoix[i]] = 1$, 其他为 0:

$$P'[i] = P[i, gtoix[i]]$$

$$\text{loss_cost} = \sum_{i=0}^{n-1} (-\ln(10^{-20} + P'[i]))$$

其中 10^{-20} 为平滑因子，使得在 $P'[i] = 0$ 时误差不至于为无穷大导致程序崩溃（不过理论上是不可能的）。

同时作者还计算了以 2 为底的对数误差：

$$\text{logppl} = \sum_{i=0}^{n-1} (-\log_2(10^{-20} + P'[i]))$$

以及记录了语句的长度，用于后面计算平均误差：

$$\text{logppln} = \text{len}(\text{gtix})$$

同时需要计算误差 loss_cost 对输出 Y 的偏导数:

$$dY = \frac{\partial \text{loss_cost}}{\partial Y}$$

因为:

$$d(\text{loss cost}) = d\left(\sum_i (-\ln(p'))\right) = d\left(\sum_i \left(-\ln\left(\frac{e^{Y'}}{\sum_j (e^{Y'})}\right)\right)\right) = d\left(\ln\left(\sum_i \sum_j e^{Y'}\right) - Y'\right)$$

所以:

$$dY[i, \text{gtoix}[i]] = \frac{\partial \text{loss cost}[i]}{\partial Y[i, \text{gtoix}[i]]} = \frac{\partial \text{loss cost}[i]}{\partial Y'[i]} = \frac{e^{Y'}}{\sum_j e^{Y'}} - 1 = P'[i] - 1$$

注意: 这里的推导省略了很多常数细节, 只是一个示意, 便于理解代码。

理论上 $dY[i, j]_{j \neq \text{gtoix}[i]} = 0$, 但在实际的代码中, 作者设置为 $dY[i, j]_{j \neq \text{gtoix}[i]} = P[i, j]$ 。

然后以上过程执行 batch_size 次, 并把每次计算得到的误差累加, 把每次计算的 dY 全部保存到 dYs 中。

2) 计算惩罚误差:

对于 misc 中的所有 regularize 参数 p , 设其对应的矩阵为 mat , 惩罚误差:

$$\text{reg cost} = \sum_{p \in \text{misc}[\text{regularize}]} (0.5 \cdot \text{regc} \cdot ||\text{mat}||^2)$$

计算得到的梯度值增加惩罚因子:

$$\text{grades}[p] = \text{grades}[p] + \text{regc} \cdot \text{mat}$$

最后会计算一个 ppl2 , 即以 2 为底的对数平均误差:

$$\text{ppl2} = \frac{\sum_{b=0}^{\text{batch_size}} \text{logppl}}{\sum_{b=0}^{\text{batch_size}} \text{logppln}}$$

以上过程的全部代码如下所示:

```
loss_cost = 0.0
dYs = []
logppl = 0.0
logppln = 0
for i, pair in enumerate(batch):
    img = pair['image']
    # ground truth indeces for this sentence we expect to see
    gtix = [wordtoix[w] for w in pair['sentence']['tokens'] if w in wordtoix]
    gtix.append(0) # don't forget END token must be predicted in the end!
    # fetch the predicted probabilities, as rows
    Y = Ys[i]
    maxes = np.amax(Y, axis=1, keepdims=True)
    e = np.exp(Y - maxes) # for numerical stability shift into good numerical range
    P = e / np.sum(e, axis=1, keepdims=True)
```

```

    loss_cost += - np.sum(np.log(1e-20 + P[range(len(gtix)),gtix])) # note: add smoothing
to not get infs
    print "shape:%d,%d" % (P[range(len(gtix)),gtix].shape[0],
P[range(len(gtix)),gtix].shape[0])
    logppl += - np.sum(np.log2(1e-20 + P[range(len(gtix)),gtix])) # also accumulate log2
perplexities
    logppln += len(gtix)

# lets be clever and optimize for speed here to derive the gradient in place quickly
for iy,y in enumerate(gtix):
    P[iy,y] -= 1 # softmax derivatives are pretty simple
    dYs.append(P)

# backprop the RNN
grads = BatchGenerator.backward(dYs, gen_caches)

# add L2 regularization cost and gradients
reg_cost = 0.0
if regc > 0:
    for p in misc['regularize']:
        mat = model[p]
        reg_cost += 0.5 * regc * np.sum(mat * mat)
        grads[p] += regc * mat

```

2.4.6.2 梯度下降的优化算法^[8]

本模型中采用的 SGD 算法，SGD 指 stochastic gradient descent，即随机梯度下降。是梯度下降的 batch 版本。

1) SGD

对于训练数据集，我们首先将其分成 n 个 batch，每个 batch 包含 m 个样本。我们每次更新都利用一个 batch 的数据，而非整个训练集。即第 t 个 batch 后的更新为：

$$\begin{aligned}\overrightarrow{\omega_{t+1}} &\leftarrow \overrightarrow{\omega_t} + \Delta \overrightarrow{\omega_t} \\ \Delta \overrightarrow{\omega_t} &= -\eta g_t\end{aligned}$$

其中， η 为学习率， g_t 为 w 在 t 时刻的梯度。这么做的好处在于：

当训练数据太多时，利用整个数据集更新往往时间上不显示。batch 的方法可以减少机器的压力，并且可以更快地收敛。

当训练集有很多冗余时（类似的样本出现多次），batch 方法收敛更快。以一个极端情况为例，若训练集前一半和后一半梯度相同。那么如果前一半作为一个 batch，后一半作为另一个 batch，那么在一次遍历训练集时，batch 的方法向最优解前进两个 step，而整体的方法只前进一个 step。

2) Momentum

SGD 方法的一个缺点是，其更新方向完全依赖于当前的 batch，因而其更新十分不稳定。解决这一问题的一个简单的做法便是引入 momentum。momentum 即动量，它模拟的是物体运动时的惯性，即更新的时候在一定程度上保留之前更新的方向，同时利用当前 batch 的梯度微调最终的更新方向。这样一来，可以在一定程度上增加稳定性，从而学习地更快，并且还有一定摆脱局部最优的能力：

$$\Delta \overrightarrow{\omega_t} = \text{momentum} \cdot \overrightarrow{\omega_t} - \eta g_t$$

其中 momentum 动量表示要在多大程度上保留原来的更新方向。

3) Adagrad

上面提到的方法对于所有参数都使用了同一个更新速率。但是同一个更新速率不一定适合所有参数。比如有的参数可能已经到了仅需要微调的阶段，但又有些参数由于对应样本少等原因，还需要较大幅度的调动。

Adagrad 就是针对这一问题提出的，自适应地为各个参数分配不同学习率的算法。其公式如下：

$$\Delta \bar{\omega}_t = -\frac{\eta}{\sqrt{\sum_{\tau=1}^t g_{\tau} + \epsilon}} g_t$$

其中 $\sum_{\tau=1}^t g_{\tau}$ 表示需要把前几次计算的梯度累加起来， ϵ 是一个防止分母为 0 的平滑因子。从该式可看出：对于每个参数，随着其更新的总距离增多，其学习速率也随之变慢。

4) Rmsprop^[9]

Adagrad 算法存在三个问题

- 其学习率是单调递减的，训练后期学习率非常小；
- 其需要手工设置一个全局的初始学习率；
- 更新 $\bar{\omega}_t$ 时，左右两边的单位不统一；

首先，针对第一个问题，我们可以只使用 adagrad 的分母中的累计项离当前时间点比较近的项，如下式：

$$E[g^2]_t = \rho E[g^2]_{t-1} + (1-\rho)g_t^2$$
$$\Delta \bar{\omega}_t = -\frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

这里 ρ 是衰减系数，通过这个衰减系数，我们令每一个时刻的 g_t 随之时间按照 ρ 指数衰减，这样就相当于我们仅使用离当前时刻比较近的 g_t 信息，从而使得还很长时间之后，参数仍然可以得到更新。

这种方法就是 Rmsprop 方法，也是本文作者在论文提到的其采用的方式。

5) Adadelta

Adadelta 针对上述三个问题都提出了比较漂亮的解决方案。首先其分母部分与 rmsprop 方法相同，解决了第一个问题。

针对第三个问题，其实 sgd 跟 momentum 系列的方法也有单位不统一的问题。sgd、momentum 系列方法中：

$$\Delta \omega \text{ 的单位} \propto g \text{ 的单位} \propto \frac{\partial f}{\partial w} \propto \frac{1}{\Delta \omega \text{ 的单位}}$$

而牛顿迭代法 $\Delta \omega = H_t^{-1} g_t$ ，其单位是正确的(f 无单位)：

$$\Delta \omega \text{ 的单位} \propto H^{-1} g \text{ 的单位} \propto \frac{\frac{\partial f}{\partial w}}{\frac{\partial^2 f}{\partial w^2}} \propto \frac{\Delta \omega \text{ 的单位}}{\partial f}$$

在解决学习率单调递减的问题的方案中，分母已经是 $\frac{\partial f}{\partial w}$ 的一个近似了。这里我们可以构造 $\Delta \omega$ 的近似，来模拟得到 H^{-1} 的近似，从而得到近似的牛顿迭代法。

因此最终采用下面的公式解决：

$$\Delta \bar{\omega}_t = -\frac{\sqrt{\sum_{\tau=1}^{t-1} \omega_{\tau}}}{\sqrt{E[g^2]_t + \epsilon}}$$

以上五种方案的优劣程度，本文没有比较，但是作者 Karpathy 进行了一些仿真的测试比较，可以参考 <http://cs.stanford.edu/people/karpathy/convnetjs/demo/trainers.html>。

2.4.6.3 step 函数

`step(batch, model, cost_function, **kwargs)` 函数输入为一批次的图像语句对 `batch`，模型参数 `model`，误差计算函数 `cost_function`，其他参数 `kwargs`。其中误差计算函数即为上一小节的 `RNNGenCost` 函数。

`step` 函数的主要功能就是执行一批次的数据训练，包括前向和反向传播以及参数更新。`step` 函数首先调用 `RNNGenCost` 函数计算得到所有参数的梯度值。首先进行梯度剪切，然后开始对这些梯度值进行优化。

如果运行参数中 `grad_clip > 0`，就做梯度剪切。梯度剪切，就是每次只更新对分类最有利的一部分梯度，忽略其他。代码中梯度剪切执行的操作就是将计算出来的梯度矩阵中小于 `-grad_clip` 和大于 `grad_clip` 的值全部做限幅处理；

`Step` 进行梯度优化的方法即为 2.4.6.2 中介绍的基本方法，包括：`vanilla`（即普通的 SGD），`momentum`，`rmsprop`，`adagrad`，`adadelata`。

下面我们介绍其优化步骤，在构造函数中我们初始化了两个缓存 `step_cache` 和 `step_cache2`。在 2.4.6.2 节中我们看到有些方法需要缓存历史时刻的 `gt`，如 `Adagrad`，有些方法需要缓存历史时刻的 `wt`，如 `Adadelata`。因此两个 `step_cache` 就是用来缓存历史数据(即前几个 `batch` 调用 `step` 时的计算数据)的。

对于 `update` 中的每一个参数 `p`，需要计算其对应的增量 `dx`：

首先确定参数中指定的 `slover` 优化方法(`vanilla`, `rmsprop`, `adagrad`, `adadelata`)(`momentum` 比较简单，是直接通过指定 `momentum` 因子来实现，如果该因子 `momentum > 0`，那么采用 `momentum`)；

1) 对于 `vanilla`

$$dx = -\text{learning_rate} \times \text{grads}[p]$$

2) 对于 `momentum`

$$dx = \text{momentum} \times \text{self.step_cache}[p] - \text{learning_rate} \times \text{grads}[p]$$

3) 对于 `rmsprop`

$$\text{self.step_cache}[p] = \text{self.step_cache}[p] * \text{decay_rate} + (1.0 - \text{decay_rate})(\text{grads}[p])^2$$

$$dx = \frac{-\text{learning_rate} \times \text{grads}[p]}{\sqrt{\text{self.step_cache}[p] + \text{smooth_eps}}}$$

4) 对于 `adagrad`

$$\text{self.step_cache}[p] = \text{self.step_cache}[p] + (\text{grads}[p])^2$$

$$dx = \frac{-\text{learning_rate} \times \text{grads}[p]}{\sqrt{\text{self.step_cache}[p] + \text{smooth_eps}}}$$

5) 对于 `adadelata`

$$\text{self.step_cache}[p] = \text{self.step_cache}[p] * \text{decay_rate} + (1.0 - \text{decay_rate})(\text{grads}[p])^2$$

$$dx = \frac{\sqrt{\text{self.step_cache2}[p] + \text{smooth_eps}}}{\sqrt{\text{self.step_cache}[p] + \text{smooth_eps}}}$$

$$\text{self.step_cache2}[p] = \text{self.step_cache2}[p] * \text{decay_rate} + (1.0 - \text{decay_rate})(\text{grads}[p])^2$$

注意：此处给定的形式与前一节的公式形式上是采用源代码来表示的，但原理是完全相同的。

此处相应的代码如下所示：

```
# perform parameter update
for p in update:
```

```

if p in grads:

    if solver == 'vanilla': # vanilla sgd, optional with momentum
        if momentum > 0:
            dx = momentum * self.step_cache[p] - learning_rate * grads[p]
            self.step_cache[p] = dx
        else:
            dx = - learning_rate * grads[p]

    elif solver == 'rmsprop':
        self.step_cache[p] = self.step_cache[p] * decay_rate + (1.0 - decay_rate) * grads[p]
** 2
        dx = -(learning_rate * grads[p]) / np.sqrt(self.step_cache[p] + smooth_eps)

    elif solver == 'adagrad':
        self.step_cache[p] += grads[p] ** 2
        dx = -(learning_rate * grads[p]) / np.sqrt(self.step_cache[p] + smooth_eps)

    elif solver == 'adadelta':
        self.step_cache[p] = self.step_cache[p] * decay_rate + (1.0 - decay_rate) * grads[p]
** 2
        dx = - np.sqrt( (self.step_cache2[p] + smooth_eps) / (self.step_cache[p] +
smooth_eps) ) * grads[p]
        self.step_cache2[p] = self.step_cache2[p] * decay_rate + (1.0 - decay_rate) * (dx **
2)

    else:
        raise Exception("solver %s not supported" % (solver, ))

# perform the parameter update
model[p] += dx

```

计算出优化后的 dx 后, 将 `model` 中的参数加上相应的增量;

`step` 函数返回计算出的误差 `cost` 和当前的平均 \log_2 误差值 `ppl2` 值, 并封装成 `json` 格式数据。

2.4.6.4 梯度检查 gradCheck 函数

2.5 模型训练流程分析

为了理清整个模型是如何展开训练的到最终对整个训练集的训练和验证的完成, 我们将从源代码的角度细细解读。因此我们关注的主要文件是 `driver.py`, 其中会调用很多其他文件的模型。我们可以从 `main` 函数开始, 展开层层调用, 直到 `main` 函数结束, 整个训练过程也结束了。

2.5.1 driver.py—__main__

```

if __name__ == "__main__":

```

```

parser = argparse.ArgumentParser()

# global setup settings, and checkpoints
parser.add_argument('-d', '-dataset', dest='dataset', default='flickr8k', help='dataset: flickr8k/flickr30k')
parser.add_argument('-a', '-do_grad_check', dest='do_grad_check', type=int, default=0, help='perform gradcheck? program will block for visual inspection and will need manual user input')
parser.add_argument('-fappend', dest='fappend', type=str, default='baseline', help='append this string to checkpoint filenames')
parser.add_argument('-o', '-checkpoint_output_directory', dest='checkpoint_output_directory', type=str, default='cv/', help='output directory to write checkpoints to')
parser.add_argument('-worker_status_output_directory', dest='worker_status_output_directory', type=str, default='status/', help='directory to write worker status JSON blobs to')
parser.add_argument('-write_checkpoint_ppl_threshold', dest='write_checkpoint_ppl_threshold', type=float, default=-1, help='ppl threshold above which we dont bother writing a checkpoint to save space')
parser.add_argument('-init_model_from', dest='init_model_from', type=str, default="", help='initialize the model parameters from some specific checkpoint?')

# model parameters
parser.add_argument('-generator', dest='generator', type=str, default='lstm', help='generator to use')
parser.add_argument('-image_encoding_size', dest='image_encoding_size', type=int, default=256, help='size of the image encoding')
parser.add_argument('-word_encoding_size', dest='word_encoding_size', type=int, default=256, help='size of word encoding')
parser.add_argument('-hidden_size', dest='hidden_size', type=int, default=256, help='size of hidden layer in generator RNNs')

# lstm-specific params
parser.add_argument('-tanhC_version', dest='tanhC_version', type=int, default=0, help='use tanh version of LSTM?')

# rnn-specific params
parser.add_argument('-rnn_relu_encoders', dest='rnn_relu_encoders', type=int, default=0, help='relu encoders before going to RNN?')
parser.add_argument('-rnn_feed_once', dest='rnn_feed_once', type=int, default=0, help='feed image to the rnn only single time?')

# optimization parameters
parser.add_argument('-c', '-regc', dest='regc', type=float, default=1e-8, help='regularization strength')
parser.add_argument('-m', '-max_epochs', dest='max_epochs', type=int, default=50, help='number of epochs to train for')
parser.add_argument('-solver', dest='solver', type=str, default='rmsprop', help='solver type: vanilla/adagrad/adadelata/rmsprop')
parser.add_argument('-momentum', dest='momentum', type=float, default=0.0, help='momentum for vanilla sgd')

```

```

parser.add_argument('--decay_rate', dest='decay_rate', type=float, default=0.999, help='decay
rate for adadelta/rmsprop')
parser.add_argument('--smooth_eps', dest='smooth_eps', type=float, default=1e-8, help='epsilon
smoothing for rmsprop/adagrad/adadelta')
parser.add_argument('-l', '--learning_rate', dest='learning_rate', type=float, default=1e-3,
help='solver learning rate')
parser.add_argument('-b', '--batch_size', dest='batch_size', type=int, default=100,
help='batch size')
parser.add_argument('--grad_clip', dest='grad_clip', type=float, default=5, help='clip
gradients (normalized by batch size)? elementwise. if positive, at what threshold?')
parser.add_argument('--drop_prob_encoder', dest='drop_prob_encoder', type=float, default=0.5,
help='what dropout to apply right after the encoder to an RNN/LSTM')
parser.add_argument('--drop_prob_decoder', dest='drop_prob_decoder', type=float, default=0.5,
help='what dropout to apply right before the decoder in an RNN/LSTM')

# data preprocessing parameters
parser.add_argument('--word_count_threshold', dest='word_count_threshold', type=int,
default=5, help='if a word occurs less than this number of times in training data, it is
discarded')

# evaluation parameters
parser.add_argument('-p', '--eval_period', dest='eval_period', type=float, default=1.0,
help='in units of epochs, how often do we evaluate on val set?')
parser.add_argument('--eval_batch_size', dest='eval_batch_size', type=int, default=100,
help='for faster validation performance evaluation, what batch size to use on val
img/sentences?')
parser.add_argument('--eval_max_images', dest='eval_max_images', type=int, default=-1,
help='for efficiency we can use a smaller number of images to get validation error')
parser.add_argument('--min_ppl_or_abort', dest='min_ppl_or_abort', type=float, default=-1,
help='if validation perplexity is below this threshold the job will abort')

args = parser.parse_args()
params = vars(args) # convert to ordinary dict
print 'parsed parameters:'
print json.dumps(params, indent = 2)
main(params)

```

从上面 main 函数的代码可以看出，整个 main 函数非常简单，即首先初始化一个参数解析器，然后添加所有支持的参数选项及其默认值。所有参数列表如下表所示。后面在使用该参数到的时候我们还会详细讲解其具体含义。

参数	默认值	功能
--dataset	flickr8k	在 data 中存在多个数据集时，训练数据集的指定
--do_grad_check	0	执行梯度计算的检查
--fappend	baseline	
--checkpoint_output_directory	cv/	指定 checkpoint 文件的输出路径
--worker_status_output_directory	status/	指定训练过程中的状态信息输出路径
--write_checkpoint_ppl_threshold	-1	指定写入 checkpoint 的 ppl2 阈值

--init_model_from	"	指定从中读取 model 参数信息的文件
--generator	lstm	指定语句生成器神经网络模型(lstm 或 rnn)
--image_encoding_size	256	指定图片的编码矢量维度
--word_encoding_size	256	指定单词的编码矢量维度
--hidden_size	256	指定 RNN 网络的隐层单元的输出维度
--tanhC_version	0	指定 LSTM 网络中是否使用正切激活函数 tanh
--rnn_relu_encoders	0	指定输入向量进入 RNN 前是否使用 RELU 激活
--rnn_feed_once	0	指定图片输入 RNN 网络时是仅一次输入还是一直输入
--regc	1e-8	指定 L2 范式惩罚因子
--max_epochs	50	指定训练的循环次数
--solver	rmsprop	指定在梯度下降时采用的优化措施，可选的有 vanilla, momentum, rmsprop, adagrad, adadelata
--momentum	0.0	指定 momentum 优化方法的动量因子
--dacay_rate	0.999	指定衰减因子
--smooth_eps	1e-8	指定平滑因子
--learning_rate	1e-3	指定学习速率
--batch_size	100	指定每批次训练的图像-语句对数目
--grad_clip	5	指定梯度剪切因子
--drop_prob_encoder	0.5	指定编码时 dropout 方法的权重不工作节点比重
--drop_prob_decoder	0.5	指定解码时 dropout 方法的权重不工作节点比重
--word_count_threshold	5	指定统计语句单词时的单词出现次数阈值
--eval_period	1.0	指定每个多少个 epoch 进行一次验证
--eval_batch_size	100	指定验证时的图片-语句对批量大小
--eval_max_images	-1	指定验证时的最大图片数量
--min_ppl_or_abort	-1	指定最小的 ppl，如果验证时计算的 ppl 小于这个值，训练将结束

设置完参数之后，__main__主程序会读取用户的输入参数，并将其以 json 文件的格式打印出来。最后主程序调用 main 函数。

2.5.2 driver.py—main()函数

main 函数实现了整个训练的过程。我们将 main 函数的代码从头到尾分为几个部分来看。

2.5.2.1 初始化参数变量

```
def main(params):
    batch_size = params['batch_size']
    dataset = params['dataset']
    word_count_threshold = params['word_count_threshold']
    do_grad_check = params['do_grad_check']
    max_epochs = params['max_epochs']
    host = socket.gethostname() # get computer hostname

    # fetch the data provider
    dp = getDataProvider(dataset)
```



```
misc = {} # stores various misc items that need to be passed around the framework
```

该部分即获取相应参数。并且初始化一个 DataProvidor。一个 DataProvidor 就是前面提到的图片文本数据读取和输出类。misc 字典结构用于存储需要在框架内传递的各种变量。

2.5.2.2 单词库预处理

```
misc['wordtoix'], misc['ixtoward'], bias_init_vector =  
preProBuildWordVocab(dp.iterSentences('train'), word_count_threshold)
```

单词库预处理的主要功能实现是 preProBuildWordVocab 功能。该函数的输入参数为：所有的训练语句，和统计单词时的出现次数阈值。其函数运行流程如下图 2-5 所示。

- 1) 筛选出出现次数不少于 5 的单词并组成单词库 vocab;
- 2) 为单词库建立单词<-->索引的双向映射，便于后期模型运算时的查找。其中需要注意的是，在论文阅读报告 3.2.3 中我们提到：“输入语句的第一个单词是一个奇怪的 START 符号，输出语句的最后一个单词是 END，这也是本文作者模型的一个关键点。因为我们需要的是输出一张图片的文本描述，因此长度是肯定有限的，因此加入了 START 为语句的开始，END 为语句的结束，首先输入 START 启动 RNN，在输出到 END 之后结束 RNN，如果是训练集，则进行参数更新，如果是测试，则输出结果。”因此此处我们定义 wordtoindex 中 #START# 的索引为 0，indextoword 中 0 索引对应结束句号“。”。
- 3) 在 2.3 节中提到 RNN 网络模型最后的输出编码为 $Y = W_d H + b_d$ ，我们会加上一个偏置向量 b_d ，这里就使用单词出现频率初始化这个偏置向量。假设单词库单词总数为 n （包括结束句号‘.’），第 i 个单词出现的次数为 C_i ，那么

$$b_d = P_i - \max_{i=0 \dots n-1} (P_i), \quad i = 0, 1, 2, \dots, n-1$$

$$P_i = \log \left(\frac{C_i}{\sum_{i=0}^n C_i} \right), \quad i = 0, 1, 2, \dots, n-1$$

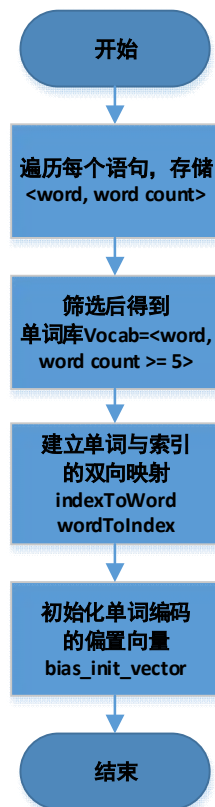


图 2-5 preProBuildWordVocab 函数运行流程图

2.5.2.3 语句生成模型的参数初始化

```
# delegate the initialization of the model to the Generator class
BatchGenerator = decodeGenerator(params)
init_struct = BatchGenerator.init(params, misc)
model, misc['update'], misc['regularize'] = (init_struct['model'], init_struct['update'],
init_struct['regularize'])

# force overwrite here. This is a bit of a hack, not happy about it
model['bd'] = bias_init_vector.reshape(1, bias_init_vector.size)

if params.get('init_model_from', ""):
    # load checkpoint
    checkpoint = pickle.load(open(params['init_model_from'], 'rb'))
    model = checkpoint['model'] # overwrite the model
```

这段代码可以看到作者生成了一个批处理的语句生成模型 `BatchGenerator`(其中 `decodeGenerator` 函数直接返回 `GenericBatchGenerator` 类)。

上面这段代码调用了 `GenericBatchGenerator` 的 `init` 方法。即会初始化整个递归神经网络的所有待训练的参数。

同时，如果在参数中指定了从文件中读取 `model` 参数的话，那么从文件中读取所有参数并赋给 `model`。因为后面会提到作者在训练的过程中会将中间结果暂存，这样如果暂停训练或者需要需要累计训练的话，可以直接从文件中恢复上次训练过程。

2.5.2.4 初始化 solver 和 costfun

然后代码会初始化一个 `Solver` 类的实例和一个误差函数 `costfun()`，即前面提到的 `RNNGenCost` 函数。

```
solver = Solver()
def costfun(batch, model):
    # wrap the cost function to abstract some things away from the Solver
    return RNNGenCost(batch, model, params, misc)
```

2.5.2.5 计算总迭代次数和误差指标

```
# calculate how many iterations we need
max_epochs = params['max_epochs']
num_sentences_total = dp.getSplitSize('train', ofwhat = 'sentences')
num_iters_one_epoch = num_sentences_total / batch_size
max_iters = max_epochs * num_iters_one_epoch
eval_period_in_epochs = params['eval_period']
eval_period_in_iters = max(1, int(num_iters_one_epoch * eval_period_in_epochs))
abort = False
top_val_ppl2 = -1
```

训练过程分为两个层级，内层是将所有训练的图像语句对(比如 flickr8 是 8000 张图片，1000 张作为测试，1000 张作为验证，6000 张用于训练，一张图片对应 5 个句子，因此一共 30000 组训练数据)按 `batch_size` 划分为很多批次，即代码中的 `num_iters_one_epoch`。一个 epoch 即代表一次 30000 组数据的训练。一个 batch 为 100 的情况下，一个 epoch 中一共有 300 个 batch，即 `num_iters_one_epoch=300`。

外层即为一共要执行多少个 epoch， 这是通过参数来指定的。默认是 50。因此总的迭代次数为：

$$\text{max_iters} = \text{max_epochs} * \text{num_iters_one_epoch}$$

另外，在训练中的过程需要进行验证，防止发生梯度爆炸等或者训练结果过拟合现象，因此还需要设置验证的迭代次数。eval_period_in_epochs 指定的是每隔多少个 epoch 验证一次，默认为 1.0。eval_period_in_iters 指定的是每隔多少个迭代周期验证一次，因此：

$$\text{eval_period_in_iters} = \max(1, \text{int}(\text{num_iters_one_epoch} * \text{eval_period_in_epochs}))$$

2.5.2.6 开始max_iters次迭代过程

每一次的迭代过程完成一批次的数据训练，其运行流程图如下所示：

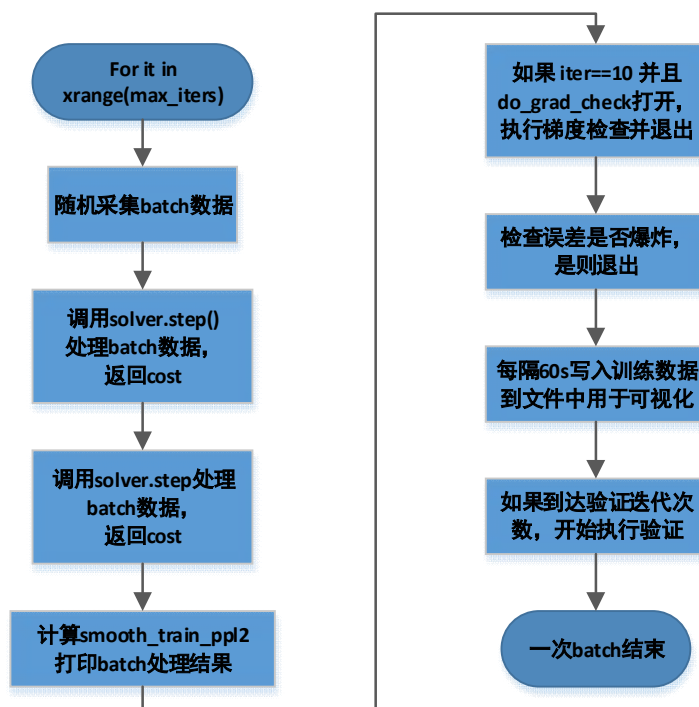


图 2-6 训练 main 函数中的迭代过程流程图

- 1) 其中前面介绍过 solver.step 函数会返回计算得到的 ppl2 误差，

$$\text{train_ppl2} = \frac{\sum_{b=0}^{\text{batch_size}} \log p_{pl}}{\sum_{b=0}^{\text{batch_size}} \log p_{pln}}$$

计算 smooth_train_ppl2 误差：

$$\text{smooth_train_ppl2} = 0.99 \cdot \text{smooth_train_ppl2} + 0.01 \cdot \text{train_ppl2}$$

smooth_train_ppl2 的目的是为了避免整体结果受单次影响太大。

- 2) 执行梯度检查的目的是为了检查作者在计算梯度的过程代码是否计算正确，避免因为简单的代码错误而导致训练资源浪费，主要是用于 debug 的。
- 3) 检查误差是否爆炸的意思就是：根据 batch 计算得到的 total_cost = loss_cost + regc_cost，如果 total_cost 比 it=0 时刻的误差的两倍还要大，那么出现误差爆炸现象，训练直接结束。
- 4) 每隔 60s 写一次 json 文件主要是为了用户可以通过网页查看其实时的训练。写入文件中 history 的内容为：

time, iter, epoch, time_per_batch, smooth_train_ppl2, val_ppl2, train_ppl2; 写入文件名为<主机名>_status.json; 其记录内容如下所示:

```
{
  "params": {
    "grad_clip": 5,
    "rnn_relu_encoders": 0,
    "dataset": "flickr8k",
    "image_encoding_size": 256,
    "eval_max_images": -1,
    "drop_prob_decoder": 0.5,
    "word_encoding_size": 256,
    "max_epochs": 50,
    "eval_batch_size": 100,
    "fappend": "baseline",
    "generator": "lstm",
    "min_ppl_or_abort": -1,
    "tanhC_version": 0,
    "write_checkpoint_ppl_threshold": -1,
    "decay_rate": 0.999,
    "rnn_feed_once": 0,
    "hidden_size": 256,
    "momentum": 0.0,
    "worker_status_output_directory": "status/",
    "init_model_from": "",
    "learning_rate": 0.001,
    "checkpoint_output_directory": "cv/",
    "do_grad_check": 0,
    "word_count_threshold": 5,
    "batch_size": 100,
    "regc": 1e-08,
    "smooth_eps": 1e-08,
    "solver": "rmsprop",
    "eval_period": 0.01,
    "drop_prob_encoder": 0.5
  },
  "history": [{
    "smooth_train_ppl2": 167.34054991880302,
    "val_ppl2": 2538,
    "train_ppl2": 167.34054991880302,
    "epoch": [0.0, 50],
    "iter": [0, 15000],
    "time": "2016-01-03T15:00:11.343000",
    "time_per_batch": 10.940000057220459
  }],
  {
    "smooth_train_ppl2": 167.19801673293205,
    "val_ppl2": 166.4841864848666,
```

```

        "train_ppl2": 157.83130308795364,
        "epoch": [0.01, 50],
        "iter": [3, 15000],
        "time": "2016-01-03T15:02:03.826000",
        "time_per_batch": 11.690999984741211
    },
    {
        "smooth_train_ppl2": 166.79243952987113,
        "val_ppl2": 132.22874411419406,
        "train_ppl2": 129.46535997912648,
        "epoch": [0.02, 50],
        "iter": [6, 15000],
        "time": "2016-01-03T15:03:59.045000",
        "time_per_batch": 12.229000091552734
    }
]
}

```

5) 验证主要就是在整个验证集上执行所有批处理的 `BatchGenerator.forward()` 函数，并计算得到累加的 ppl2 误差 `val_ppl2`，如果 `val_ppl2 > min_ppl_or_abort`，训练中止。

(注意: `min_ppl_or_abort` 是在参数中指定的，参数中说的如果 `val_ppl2 < min_ppl_or_abort`，那么训练中止。我个人感觉有点问题，因为误差是越小越好，这里我感觉应该设计 `max_ppl_or_abort`，就是最大能容忍的 ppl2，如果 `val_ppl2 > max_ppl_or_abort`，那么误差可能爆炸，训练中止，实际代码中写的也是大于，如下所示：

```

# abort training if the perplexity is no good
min_ppl_or_abort = params['min_ppl_or_abort']
if val_ppl2 > min_ppl_or_abort and min_ppl_or_abort > 0:
    print 'aborting job because validation perplexity %f < %f' % (val_ppl2,
min_ppl_or_abort)
    abort = True # abort the job
)

```

同时，在验证时，我们永远只记录**变好的训练结果**。即记录初始的 `val_ppl2` 为 `top_ppl2`，如果后续的验证 `val_ppl2` 比 `top_ppl2` 小，那么就可以记录下来，如果比其差，直接丢弃结果。在记录时，还设置了一个 `write_checkpoint_ppl_threshold`，如果 `val_ppl2` 也比这个阈值小，我们就将训练和验证的结果记录下来，记录的内容为: `iter, epoch, model, params, perplexity, wordtoix, ixtoword`。写入的文件名为:

`cv/model_checkpoint_<dataset>_<host>_<baseline>_<ppl2>.p`，使用 python 标准库的 `cPickle` 包，将上述记录内容组成的字典结构以二进制方式写入文件。

注意：checkpoint 的文件名包含 `ppl2`，说明对每一个 `ppl2`，都是新建一个文件，最终我们选择 `ppl2` 最小的文件即可。

最终，这样迭代 `max_iters` 次之后，整个训练过程结束，训练得到的所有参数都保存在 `checkpoint` 文件中。

2.6 模型预测过程分析

预测过程与训练过程的不同之处在于，输入的只有图片矢量，输出的是对于这张图片的语句描述。

模型预测分为两种：一种是直接从测试集中选取图片进行预测；一种是选用自己的图片进行预测；

2.6.1 测试集图片预测

同样的，为了理清整个模型在测试时是如何工作的，我们将从源代码来解读文件 `eval_sentence_predictions.py`，从 `main` 函数开始，直到 `main` 函数结束。

2.6.1.1 `eval_sentence_predictions.py`—`__main__`

```
if __name__ == "__main__":

    parser = argparse.ArgumentParser()
    parser.add_argument('checkpoint_path', type=str, help='the input checkpoint')
    parser.add_argument('-b', '--beam_size', type=int, default=1, help='beam size in inference. 1
indicates greedy per-word max procedure. Good value is approx 20 or so, and more = better.')
    parser.add_argument('--result_struct_filename', type=str, default='result_struct.json',
help='filename of the result struct to save')
    parser.add_argument('-m', '--max_images', type=int, default=-1, help='max images to use')
    parser.add_argument('-d', '--dump_folder', type=str, default="", help='dump the relevant
images to a separate folder with this name?')

    args = parser.parse_args()
    params = vars(args) # convert to ordinary dict
    print 'parsed parameters:'
    print json.dumps(params, indent = 2)
    main(params)
```

同样的，`__main__` 函数也是先处理参数。支持的参数如下：

参数	默认值	功能
<code>checkpoint_path</code>	无	指定训练得到的 <code>checkpoint</code> 文件路径
<code>--beam_size</code>	1	指定 <code>beam</code> 搜索的 <code>size</code> 。 (待补充)
<code>--result_struct_filename</code>	<code>result_struct.json</code>	指定输出结果的文件名
<code>--max_images</code>	-1	指定最多使用的图片数量
<code>--dump_folder</code>	""	指定后预测时会将测试图片拷贝到该文件夹

然后调用 `main()` 函数处理。

2.6.1.2 `eval_sentence_predictions.py`—`main()` 函数

`main` 函数执行整个在测试数据的 `test` 集上的训练，主要分为以下几个步骤：

1) 读取 `checkpoint` 数据

在训练的时候最终会通过 `cPickle` 将训练的结果写入到 `checkpoint` 文件中，因此在预测时，首先需要从其中读取参数信息，包括 `params`, `dataset`, `model`, `wordtoindex`, `indextoword`。

2) 建立图片 `dump` 输出文件夹

如果在参数中指定了 `--dump_folder`，那么如果该文件夹不存在的话，需要建立该文件夹。

3) 初始化 `dataProvider` 和 `BatchGenerator`

`dataProvider` 用于从测试集中读取图片数据和基准输出数据，`BatchGenerator` 用于执行 `predict` 过程计算输出结果。

4) 对于测试集中的每一张图片，执行以下流程：

其中，因为我们对一张图片调用的是批处理的 `predict`，因为返回的 `Ys` 是只有一个元素的，这个元素是一个数组，数组中的每一项为<误差，预测语句单词序列>，我们取数组中的第一项为预测最准确的结果。

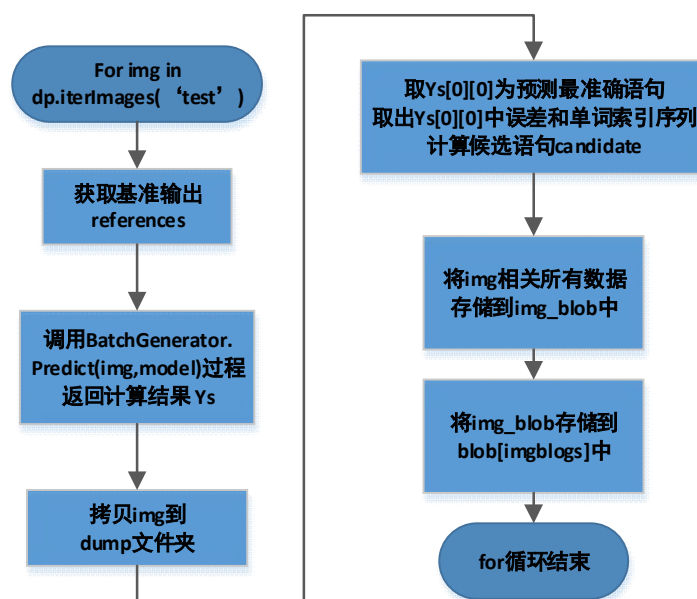


图 2-7 预测时每张图片的处理流程

5) 调用 `eval` 目录下的 `multi-bleu.perl` 脚本执行 BLEU 分数计算

经过上一步的 `for` 循环后，得到了对每一张图片的五个 `references` 语句和一个 `candidate` 语句，首先将这些信息写入文件，即先将所有 `candidate` 按行写入 `eval/output` 文件中，再将所有 `references` 分别写入 `reference_1` 到 `reference_5` 这 5 个文件中（因为每个图片对应 5 个 `reference`），最后使用 `perl` 脚本来计算 BLEU 得分

6) 计算测试数据 `test` 集上的 `ppl2` 误差计算，并写入 `blob` 中；

7) 最终将所有的预测结果组成的 `bolb` 字典存储为 `json` 格式文件，文件名由参数 `result_struct_filename` 指定。

2.6.2 自定义图片预测

自定义图片预测与测试集图片的预测代码上基本完全相同，唯一不同点在于，需要在参数中指定 `root_path`。而且 `root_path` 中需要包括以下文件：

- 1) 所有 `Image` 图片文件；
- 2) `tasks.txt` 文件，其中每一行都是待预测的图片文件名称；
- 3) 使用 `caffe` 生成的图像矢量信息文件：`vgg_feats.mat`。具体生成方式请参考 2.2.2 节。

3. 源代码运行实验

3.1 源代码的运行环境和数据准备

3.1.1 运行环境

实验源代码采用的是 `python` 语言编写，其中用到的库有：

- 1) **numpy**: 一个科学计算库。支持高阶大量的维度阵列与矩阵运算，此外也针对矩阵运算提供大量的数学函数库。
- 2) **scipy**: 一个开源的 Python 算法库和数学工具包。SciPy 包含的模块有最优化、线性代数、积分、插值、特殊函数、快速傅里叶变换、信号处理和图像处理、常微分方程求解和其他科学与工程中常用的计算。
- 3) **argparse**: argparse 模块使得编写用户友好的命令行接口非常容易。程序只需定义好它要求的参数，然后 argparse 将负责如何从 sys.argv 中解析出这些参数。argparse 模块还会自动生成帮助和使用信息并且当用户赋给程序非法的参数时产生错误信息。

可以选择一个个的安装这个 python 库，也可以选择一些已经打包好的用于机器学习的集成 python 开发环境，如 Anaconda(<https://www.continuum.io>)。

3.1.2 数据准备

需要的数据为：图片测试集数据包括 vgg_feats.mat，dataset.json 以及数据集对应的图片文件，可以从作者网站上下载。将下载后的文件放在 data 文件夹下，如下图所示：

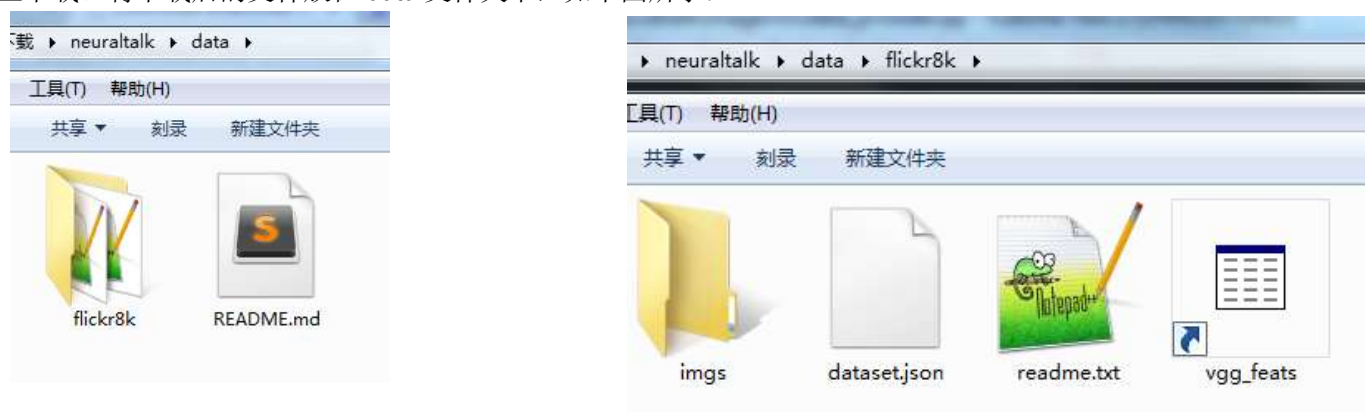


图 3-1 数据集文件及其对应文件示意图

3.2 模型训练

3.2.1 启动训练

安装好上述开发环境后，可以开始训练模型。

在项目的根路径下，执行以下代码，可以开始训练模型。

```
python driver.py
```

以上训练时是以默认的配置运行，如使用 LSTM 模型，rmsprop 优化算法等，可以在命令行指定其他参数：

```
python driver.py -generator rnn -solver adadelta
```

训练开始后会得到如下输出：首先打印出参数，然后打印出 model 要训练的所有参数，然后开始训练，每一批次即为一次迭代，打印一次训练结果。


```
[jkryan@lab36951 neuraltalk2]$ python driver.py --generator rnn
parsed parameters:
{
  "grad_clip": 5,
  "rnn_relu_encoders": 0,
  "dataset": "flickr8k",
  "image_encoding_size": 256,
  "eval_max_images": -1,
  "drop_prob_decoder": 0.5,
  "word_encoding_size": 256,
  "max_epochs": 50,
  "eval_batch_size": 100,
  "fappend": "baseline",
  "generator": "rnn",
  "min_ppl_or_abort": -1,
  "tanhC_version": 0,
  "write_checkpoint_ppl_threshold": -1,
  "decay_rate": 0.999,
  "rnn_feed_once": 0,
  "hidden_size": 256,
  "momentum": 0.0,
  "worker_status_output_directory": "status/",
  "init_model_from": "",
  "learning_rate": 0.001,
  "checkpoint_output_directory": "cv/",
  "do_grad_check": 0,
  "word_count_threshold": 5,
  "batch_size": 100,
  "regc": 1e-08,
  "smooth_eps": 1e-08,
  "solver": "rmsprop",
  "eval_period": 1.0,
  "drop_prob_encoder": 0.5
}
Initializing data provider for dataset flickr8k...
BasicDataProvider: reading data/flickr8k/dataset.json
BasicDataProvider: reading data/flickr8k/vgg_feats.mat
preprocessing word counts and creating vocab based on word count threshold 5
filtered words from 7374 to 2537 in 1.32s
model init done.
model has keys: bd, be, We, Wd, bhh, Whh, bxh, Ws, Wxh
updating: We [4096x256], be [1x256], Ws [2538x256], Whh [256x256], bhh [1x256], Wxh [256x256], bxh [1x256], Wd [256x2538], bd [1x2538]
updating: We [4096x256], Ws [2538x256], Whh [256x256], Wxh [256x256], Wd [256x2538]
number of learnable parameters total: 2482410
0/15000 batch done in 49.910s. at epoch 0.00. loss cost = 64.177309, reg cost = 0.000000, ppl2 = 242.24 (smooth 242.24)
```

图 3-2 训练结果输出截图

训练是单线程的，训练时间根据机器各有差异，我在没有 GPU 的情况下，在自己笔记本上(core-i3,4G)需要 10 多秒一个 batch，在服务器(32 核，至强 2Ghz,64G 内存)上需要 40-80 秒一个 batch。作者说他采用 GPU 的情况下是 1 秒一个 batch。

整个训练过程的最终结果将写入 checkpoint 文件中。注意每一次验证的较优结果都会写入一个文件，因此我们最终选取 ppl2 最小的一个 checkpoint 文件即可。

注意：默认的参数设置是为 LSTM 模型设置的，就是说不指定任何参数的情况下，可以得到训练结果，但是如果用默认参数来训练 RNN，会出现误差发生了爆炸，因此，**RNN 需要手动指定参数**，当然我还没弄明白什么情况下是最优的，作者也没有给出他的 RNN 参数设置，我的一个设置如下所示，主要是要调低学习率：

```
python driver.py --generator rnn --image_encoding_size 512 --word_encoding_size 512
--hidden_size 512 --momentum 0.5 --drop_prob_encoder 0.2 --drop_prob_decoder 0.2 --
learning_rate 1e-4
```

3.2.2 网页显示训练过程

训练过程中，每隔 60s 会写一次本地文件 status/<主机名>_status.json。

首先我们可以使用 python 在项目根目录下启动一个简单的 web 服务器，监听 8123:

```
python -m SimpleHttpServer 8123
```

然后访问 <http://localhost:8123/monitorcv.html>，这样监控的 html 文件 monitorcv.html 会去读取这个本地文件，

并且训练结果显示在网页上如下图所示。

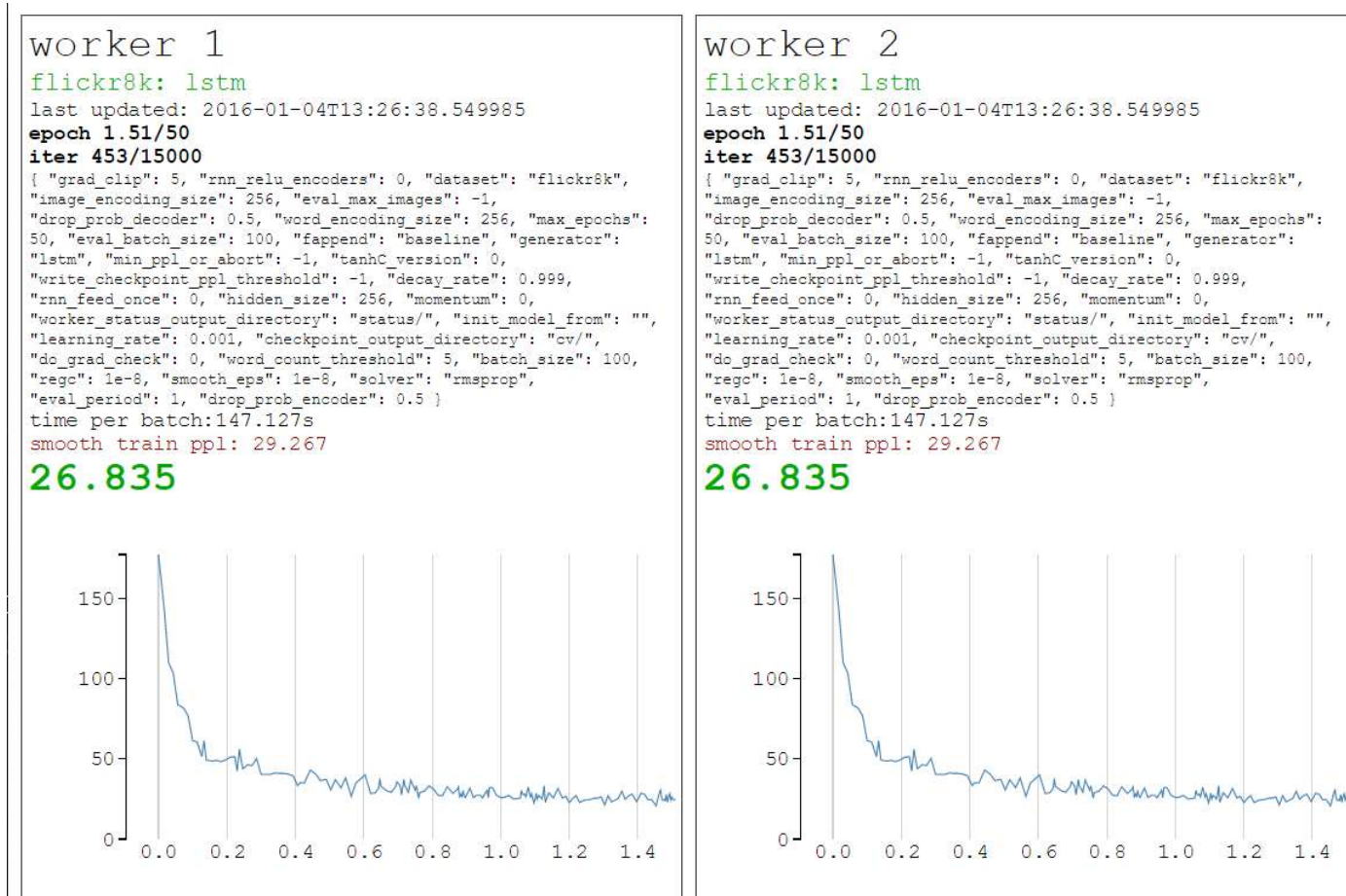


图 3-3 网页显示训练过程示意图

其中 worker 的概念是因为作者在训练的时候是在他们实验室的集群上进行训练，多个 worker 同时进行训练，每个 worker 都会写入一个 status/<主机名>_<worker id>_status.json。，这样可以网页上可以同时显示多个 worker 的训练过程。

3.3 模型预测

3.3.1 启动预测

3.3.1.1 Test 集数据预测

如果是预测测试数据中的 test 图片，那么只需要按训练过程一样准备好测试集的数据，包括(imgs 文件夹图片，vgg_feats.mat, dataset.json, 全部放在 data 文件夹下相应的 dataset 文件夹中)，准备好在此 dataset 下已经训练好的 checkpoint 文件放在 cv 文件夹下，然后在项目根目录下调用以下命令：

```
python ./eval_sentence_predictions.py ./cv/flickr8k_cnn_lstm_v1.p --beam_size 2 --  
result_struct_filename result_struct.json --max_images -1 --dump_folder ""
```

其中参数可以自由指定，后程序开始执行，并输出结果如下：

```
[jkyan@lab36951 neuraltalk2]$ python ./eval_sentence_predictions.py ./cv/flickr8k_cnn_lstm_v1.p --beam_size 2 --result_struct_filename result_struct.json --max_images -1 --dump_folder ""
parsed parameters:
{
  "result_struct_filename": "result_struct.json",
  "beam_size": 2,
  "checkpoint_path": "./cv/flickr8k_cnn_lstm_v1.p",
  "dump_folder": "",
  "max_images": -1
}
loading checkpoint ./cv/flickr8k_cnn_lstm_v1.p
Initializing data provider for dataset flickr8k...
BasicDataProvider: reading data/flickr8k/dataset.json
BasicDataProvider: reading data/flickr8k/vgg_feats.mat
image 1/-1:
GT: the dogs are in the snow in front of a fence
GT: the dogs play on the snow
GT: two brown dogs playfully fight in the snow
GT: two brown dogs wrestle in the snow
GT: two dogs playing in the snow
PRED: (-5.054593) two dogs are playing in the snow
image 2/-1:
GT: a brown and white dog swimming towards some in the pool
GT: a dog in a swimming pool swims toward somebody we cannot see
GT: a dog swims in a pool near a person
GT: small dog is paddling through the water in a pool
GT: the small brown and white dog is in the pool
PRED: (-12.343400) a little girl is playing with a red ball on the grass
image 3/-1:
GT: a man and a woman in festive costumes dancing
GT: a man and a woman with feathers on her head dance
GT: a man and a woman wearing decorative costumes and dancing in a crowd of onlookers
GT: one performer wearing a feathered headdress dancing with another performer in the streets
GT: two people are dancing with drums on the right and a crowd behind them
PRED: (-9.556779) a man and a woman are posing for a picture
```

图 3-4 test 集预测程序输出结果

即首先打印出参数，然后读取 checkpoint 和 dataset，然后对测试中的每一张图片，打印出基准描述和预测描述，及相应的 ppl2 误差。全部结束后会计算 BLEU 结果，如下图所示。

```
image 999/-1:
GT: a man who has a gray beard and gray hair laughs while wearing a purple shirt
GT: a man with a bushy beard and pink shirt is smiling
GT: a man with disheveled hair and facial hair in a crowd of people is smiling
GT: a man with poofy gray hair laughing
GT: an old man with white long hair and a white long beard
PRED: (-15.580970) a man in a red shirt and a woman in a white dress
image 1000/-1:
GT: a big dog stands on his hand leg as tennis balls are thrown his direction
GT: a brown and white dog in front of a shed overwhelmed by the onslaught of tennis balls
GT: a brown and white dogs stands in front of a wooden building while tennis balls fly through the air
GT: a dog jumps for several tennis balls thrown at him
GT: a dog stands on his hind legs amid a shower of tennis balls
PRED: (-7.619412) a black and white dog is jumping over a hurdle
writing intermediate files into eval/
invoking eval/multi-bleu.perl script...
BLEU = 57.2/38.6/25.4/16.8 (BP=1.000, ratio=1.022, hyp_len=10489, ref_len=10265)
evaluating test performance in batches of 100
evaluated 5000 sentences and got perplexity = 15.687797
perplexity of ground truth words based on dictionary of 2538 words: 15.687797
saving result struct to result_struct.json
```

图 3-5 测试数据 test 集预测结果截图 2

3.3.1.2 自定义图片预测

如果是预测自定义的图片，那么需要的数据为：所有待预测的图片文件，tasks.txt 文件（其中每一行都是待预测的图片文件名称），使用 caffe 生成的图像矢量信息文件：vgg_feats.mat。并且准备好在此 dataset 下已经训练好的 checkpoint 文件放在 cv 文件夹下，然后在项目根目录下调用以下命令：

```
python ./predict_on_images.py ./cv/flickr8k_cnn_lstm_v1.p --root_path
example_images --beam_size 2
```

其中参数可以自由指定，后程序开始执行，并输出结果如下：

```
[jkyan@lab36951 neuraltalk2]$ python ./predict_on_images.py ./cv/flickr8k_cnn_lstm_v1.p --root_path example_images --beam_size 2
parsed parameters:
{
  "beam_size": 2,
  "checkpoint_path": "./cv/flickr8k_cnn_lstm_v1.p",
  "root_path": "example_images"
}
loading checkpoint ./cv/flickr8k_cnn_lstm_v1.p
image 0/16:
PRED: (-12.227388) a white dog with a red collar is running through a field
image 1/16:
PRED: (-7.035189) a brown dog is running through a field
image 2/16:
PRED: (-7.452632) a brown dog is running through the grass
image 3/16:
PRED: (-8.255932) a dog sits on its hind legs
image 4/16:
PRED: (-14.273952) a man in a black shirt is standing in front of a white building
image 5/16:
PRED: (-11.523479) a brown dog is laying on a white carpet
image 6/16:
PRED: (-11.160877) a small dog is sitting on a tree stump
image 7/16:
PRED: (-11.486791) a child in a red jacket is standing in the snow
image 8/16:
PRED: (-17.440097) a man in a white shirt and a white shirt is playing a guitar
image 9/16:
PRED: (-15.266754) a man in a black shirt and a woman in a white shirt
image 10/16:
PRED: (-7.180625) a group of people are posing for a picture
image 11/16:
PRED: (-19.250958) a man in a black shirt and a white hat is sitting on a bench
image 12/16:
PRED: (-8.609164) a man and a woman sit on a bench
image 13/16:
PRED: (-13.661313) a man in a blue shirt is sitting on a rock
image 14/16:
PRED: (-9.501791) a group of people are gathered in a line
image 15/16:
PRED: (-11.765277) a man and a woman are sitting at a table in a restaurant
writing predictions to example_images/result_struct.json...
writing html result file to example_images/result.html...
```

图 3-6 自定义图片预测输出截图

其中，首先打印参数结果，然后载入 checkpoint，然后会输出当前图片下的所有预测结果及其误差值。同时会将结果输出到 result_struct.json 中，还会产生一个网页显示的文件。

3.3.2 网页显示预测结果

自定义图片的网页显示结果如下：



(-12.227388) a white dog with a red collar is running through a field



(-7.035189) a brown dog is running through a field

图 3-7 自定义图片预测的网页显示结果

3.4 部分预测结果

下面是我自己的部分测试结果。

准备两个 checkpoint，分别是在 flickr8 和 MSCOCO 上训练好的 LSTM 模型(这两个都是作者在其网站上提供的训练的比较好的模型，)：

- 1) Flickr8k_cnn_lstm_v1_15.69.p
- 2) Coco_cnn_lstm_v2_11.56.p

准备 12 张图片其中 4 张来自 flickr8k 中的 train(1-4)，4 张来自 flickr8k 中的 test(5-8)，4 张自定义(9-12s)。

3) 分别执行模型(1)(2)对 12 张图片的预测，结果如下：

(1)代表 Flickr8k_cnn_lstm_v1_15.69.p 的预测结果，(2) 代表 Coco_cnn_lstm_v2_11.56.p 的预测结果。



[1] (-12.557500) a basketball player in a red uniform is playing with a ball

[2] (-6.361789) a baseball player holding a bat on top of a field

A basketball player performing a lay-up;

A boy in a blue basketball uniform , number 13 and a boy in a white basketball uniform , number 23 jump for the ball;

A man in a white uniform jumps while holding a basketball as another in blue blocks him;

Basketball player wearing a white , number 23 jersey jumps up with the ball while guarded by number 13 on the opposite team ;

The man in white is playing basketball against the man in blue;



[1] (-12.138232) a man in a white shirt and sunglasses is smiling

[2] (-7.284362) a woman holding a cell phone to her ear

a large woman puts her sunglasses on .

A woman in a black dress holding sunglasses .

a woman wearing a black and white outfit while holding her sunglasses .

The lady is holding her glasses .

The woman with blond hair holds her sunglasses .



[1] (-4.635422) two dogs are running through the snow



[1] (-5.649940) a dog is running through the snow

a brown and white dog is sniffing the snow covered ground .

[2] (-8.252633) a black and white dog standing on top of a sandy beach

A brown and white dog walking in the snow near a fence .

A brown & white greyhound dog sniffs the snow .

A lean white and brown dog makes its way through a patch of snow .

A slender dog checks out a snowy patch of land .

[2] (-8.408086) a person on a snowboard in the snow
A golden retriever standing outside in the snow with a person standing with skis and poles .

A man and a dog outside in a snow storm .

A person is standing on skis in the snow with a dog .

Person with skis and a dog are standing in the snow .

The man is skiing in the snow with a large brown dog .



[1] (-7.101255) a white dog is running through a field

[2] (-8.823180) a black and white dog standing on top of a field

Two white dogs are running on the grass .

Two white dogs are running through the grass .

Two white dogs running in a field

two white dogs running through the grass

Two white dogs with cutoff tails running in green grass .



[1] (-18.300682) a man wearing a black hat and black hat is standing in front of a white building

[2] (-10.218569) a man in a suit and tie standing next to a bus

A woman and two elderly men are standing in front of a speaker .

A woman and two men wearing black jackets .

A woman and two men wearing hats and glasses are standing under a avrovulcan.com sign .

A woman and two older gentlemen who are both wearing hats .

Three older people stand in front of an avrovulcan.com sign .



[1] (-7.920553) a young boy in a swimming pool





[2] (-9.060566) a man is riding a wave on a surfboard

A kid wearing a life jacket in a swimming pool .



[1] (-9.603902) a group of people are standing in a field

[2] (-6.831159) a group of people standing on top of a sandy beach

<p>A young child in a pool wearing a life vest . a young child swimming in a pool wearing a blue life jacket Two black children , one wearing a floatation device , play in a pool . Two boys in a swimming pool</p>	<p>A group of dark-skinned people are walking past barbed wire . A group of people are walking down a road passing by some barb-wire . A group of people carrying bags walk near a barb wire fence . A poor family is leaving their home with only a few belongings . A small group of black adults and children are walking on a brown surface .</p>
 <p>[1] (-11.523479) a brown dog is laying on a white carpet [2] (-10.288552) a small dog is sitting on a toilet</p>	 <p>[1] (-15.266754) a man in a black shirt and a woman in a white shirt [2] (-13.541685) a man in a suit and tie is standing in front of a window</p>
 <p>[1] (-7.180625) a group of people are posing for a picture [2] (-6.174398) a group of people standing next to each other</p>	 <p>[1] (-11.765277) a man and a woman are sitting at a table in a restaurant [2] (-6.934166) a man standing in front of a refrigerator</p>

其中，红色是模型 1 预测的，蓝色是模型 2 预测的，绿色是基准翻译。从中看不出很明显的规律，但是在 flickr8 上训练的模型对 flickr8 中图片的表现比在 coco 上训练的模型的表现要好。对自定义图片的表现，则是 coco 表现要好，因为 coco 训练模型的数据集基数要大很多。

4. 感想和总结

通过本次对论文代码的阅读和运行实现，让我对神经网络的具体实现、训练和预测过程有了更深的了解。在看代码的过程中，发现作者的代码结构非常整齐，很多函数或类的定义都与 caffe 中的定义类似，如 forward, solver

等，这也说明神经网络的代码结果很多结构都是很相似的，重要的过程是在各个参数的选择和优化上，这些都是值得我学习。另外作者的开源精神值得我学习，非常详细的注释给我作为一个新手的入门提供了很大的帮助。也非常感谢《机器学习》课程给我提供的这样的机会来对顶尖国际会议上的论文及其实现能有完整的理解，这对我们的科研生涯非常有帮助。

5. 参考文献

- [1] Karpathy A, Fei-Fei L. Deep visual-semantic alignments for generating image descriptions[J]. arXiv preprint arXiv:1412.2306, 2014.
- [2] karpathy/neuraltalk on Github, <https://github.com/karpathy/neuraltalk>, 2015. Accessed: 2015-10-18.
- [3] K. Simonyan, A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv technical report, 2014
- [4] Caffe Zoo, <https://github.com/BVLC/caffe/wiki/Model-Zoo#models-used-by-the-vgg-team-in-ilsvrc-2014>, 2014. Accessed: 2015-12-18.
- [5] Vinyals O, Toshev A, Bengio S, et al. Show and Tell: A Neural Image Caption Generator[C]//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2015: 3156-3164.
- [6] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In NIPS, 2013.
- [7] Deep learning: 四十一(Dropout 简单理解), <http://www.cnblogs.com/tornadomeet/p/3258122.html>, 2013. Accessed: 2015-11-3.
- [8] 各种优化方法总结比较 (sgd/momentum/Nesterov/adagrad/adadelata), <http://blog.csdn.net/luo123n/article/details/48239963>, 2015. Accessed: 2015-11-3.
- [9] T. Tieleman and G. E. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. 2012.