# 1.Vocabulary

**category:**大类的字典，每句话都会针对某事物描述，该事物即大类的意义

**featCate:**类，同value组成kv对，即featVal,添加有两个特殊符"<MARKER>", "<SENT>"

**type:**类，一般单独出现，和featCate的值一样，特殊符不同，"<GENERAL>"

**featVal:**值，同key组成kv对，即featCate.，添加特殊符"<S>", "<ADJ>"。

上述几个属性均有id2xx和xx2id的值。

然后，

**id2word:**句子的词表，由id到word，㓟三个特殊符"<S>", "</S>","<UNK>",存在对应的**word2id**

**id2vec:**句子的词表，由id到word\_vec，word\_vec是由预训练好的文件获得的。

将featCate的相关词存在前面,从而使查找的时候快速寻到。

keywords\_cnt:词表的大小

# 2.Config

各种配置，包含有

1.结果存储地址

含有结果，最好模型，临时模型的地址。

2.各种输入地址

含有原始数据，预训练词向量数据，词表数据，整理好的训练数据。

## 3.数据集信息

word\_dim:词向量的纬度

bucket\_width:为桶排序的宽度，为方便处理，对数据集进行了桶排序，将group\_cnt相同的数据存在一起。 group\_cnt代表的是一句话多少个seg。

num\_buckets：最大的数，即超过num\_buckets的seg数的句子都归为一类

shuffle\_buffer\_size:随机缓冲区

## 4.训练信息

epoch:迭代次数

num\_training\_step:用于控制学习率变化的参数，是训练的step数

train\_batch\_size:训练时的batch大小

test\_batch\_size:测试时的batch大小

steps\_per\_stat:多少个step记录一次状态

# 3.Dataset

确认输入参数，针对一句话，即一个输入有如下参数：

## 原始数据：

{

"feature": [["类型", "裤"], ["版型", "宽松"], ["风格", "性感"], ["图案", "线条"], ["裤型", "阔腿裤"]],

"title": "",

"largeSrc": "http://gw.alicdn.com/imgextra/i1/697022431/TB2G92uehWYBuNjy1zkXXXGGpXa\_!!697022431-0-daren.jpg\_790x10000Q75.jpg",

"refSrc": "https://market.m.taobao.com/apps/market/content/index.html?&contentId=200569599939",

"desc": "宽松 的 阔腿裤 这 两年 真的 吸粉 不少 ， 明星 时尚 达人 的 心头 爱 。 毕竟 好 穿 时尚 ， 谁 都 能 穿 出腿长 2 米 的 效果 宽松 的 裤腿 ， 当然 是 遮肉 小 能手 啊 。 上身 随性 自然 不 拘束 ， 面料 亲肤 舒适 贴身 体验 感 棒棒 哒 。 系带 部分 增加 设计 看点 ， 还 让 单品 的 设计 感更强 。 腿部 线条 若隐若现 的 ， 性感 撩人 。 颜色 敲 温柔 的 ， 与 裤子 本身 所 呈现 的 风格 有点 反差 萌 。",

"file": "4e9cf852962fec5119cdcffa5d9c1293.jpg",

"专有属性": [["裤型", "阔腿裤"], ["类型", "裤"]],

"共有属性": [["版型", "宽松"], ["风格", "性感"], ["图案", "线条"]],

"segment": {

"seg\_0": {

"segId": 0,

"key\_type": ["裤型", "版型"],

"order": [["版型", "宽松"], ["裤型", "阔腿裤"]],

"seg": "宽松 的 阔腿裤 这 两年 真的 吸粉 不少 ， 明星 时尚 达人 的 心头 爱 。"},

"seg\_1": {

"segId": 1,

"key\_type": ["裤型", "版型"],

"order": [["版型", "宽松"]],

"seg": "毕竟 好 穿 时尚 ， 谁 都 能 穿 出腿长 2 米 的 效果 宽松 的 裤腿 ， 当然 是 遮肉 小 能手 啊 。"},

"seg\_2": {

"segId": 2,

"key\_type": ["材质"],

"order": [],

"seg": "上身 随性 自然 不 拘束 ， 面料 亲肤 舒适 贴身 体验 感 棒棒 哒 。"},

"seg\_3": {

"segId": 3,

"key\_type": ["<GENERAL>"],

"order": [],

"seg": "系带 部分 增加 设计 看点 ， 还 让 单品 的 设计 感更强 。"},

seg\_4": {

"segId": 4,

"key\_type": ["图案", "风格"],

"order": [["图案", "线条"], ["风格", "性感"]],

"seg": "腿部 线条 若隐若现 的 ， 性感 撩人 。"},

"seg\_5": {

"segId": 5,

"key\_type": ["颜色", "风格"],

"order": [],

"seg": "颜色 敲 温柔 的 ， 与 裤子 本身 所 呈现 的 风格 有点 反差 萌 。"}}

}

## 输入数据（1条）

根据上述的一条原始数据，得到如下输入

**seq\_kv\_num**:sequence中的kv对的数量

**seg\_num**:句子中seg的数量

**seg\_i\_word\_num**:句子中第i个seg的词数量。

**seg\_kv\_num**:句子中seg中的kv对的数量。

**seg\_type\_num**:句子中seg中的type的数量

**seq\_word\_num**:句子的词数量

key\_input:

list,类，是**整个句子**中kv对的k部分，取决于原始数据中的feature所表示的第一部分。**[seq\_kv\_num,]**

val\_input:

list,值,是整个句子中kv对的v部分，取决于原始数据中的feature所表示的第二部分。**[seq\_kv\_num,]**

input\_lens:

大小，是整个句子中kv对的数量，取决于原始数据中的feature所表示的所有数量。**seq\_kv\_num**

target\_input：

句子中每个seg的句子，取决于原始数据中segment部分中每个seg\_i中的seg部分的句子，并在前面加上开始符。 segment->seg\_i->seg。 **[seg\_num, seg\_i\_word\_num]**

target\_output:

句子中每个seg的值，取决于原始数据中segment部分中每个seg\_i中的seg部分的句子，并在后面加上结束符。 segment->seg\_i->seg。 **[seg\_num, seg\_i\_word\_num]**

output\_lens：

大小，句子中每个seg的长度的记录，取决于原始数据中segment部分中每个seg\_i中的seg部分的句子的长度. **[seg\_num,]**

group:

记录了每个seg都有哪些kv对，用kv对的编号进行记录，该seg有哪些kv对取决于原始数据segment部分中每个seg\_i中的order部分，转换的id取决于feature部分的第几个kv对。**[seg\_num, seg\_kv\_num]**

group\_lens:

记录了每个seg的kv对的数量，该seg有哪些kv对取决于原始数据segment部分中每个seg\_i中的order部分 **[seg\_num.]**

group\_cnt**:**

记录有多少个seg。**seg\_num**.

target\_type:

记录每个seg都描述了什么类，取决于原始数据中segment部分中每个seg\_i中的key\_type部分。 **[seg\_num, seg\_type\_num]**

target\_type\_lens：

记录每个seg描述的类的数量，**[seg\_num,]**

text:

整个句子的id形式。取决于原始数据的desc部分。 **[seq\_word\_num,]**

slens:

整个句子的长度， **seq\_word\_num**

category:

属于什么大类，取决于原始数据的feature中的类型kv对， **cate\_id**

## 注意

1.其中category每个句子只有一个，且在feature的描述中必有

2.seg中的key\_type和order，两个的类并不一致，会存在key\_type有而order没有的情况

3.order的kv对的获取，取决于seg中有无词典中v的描述，如果只有k是不能得到order中的kv对的，但是只有v就可。

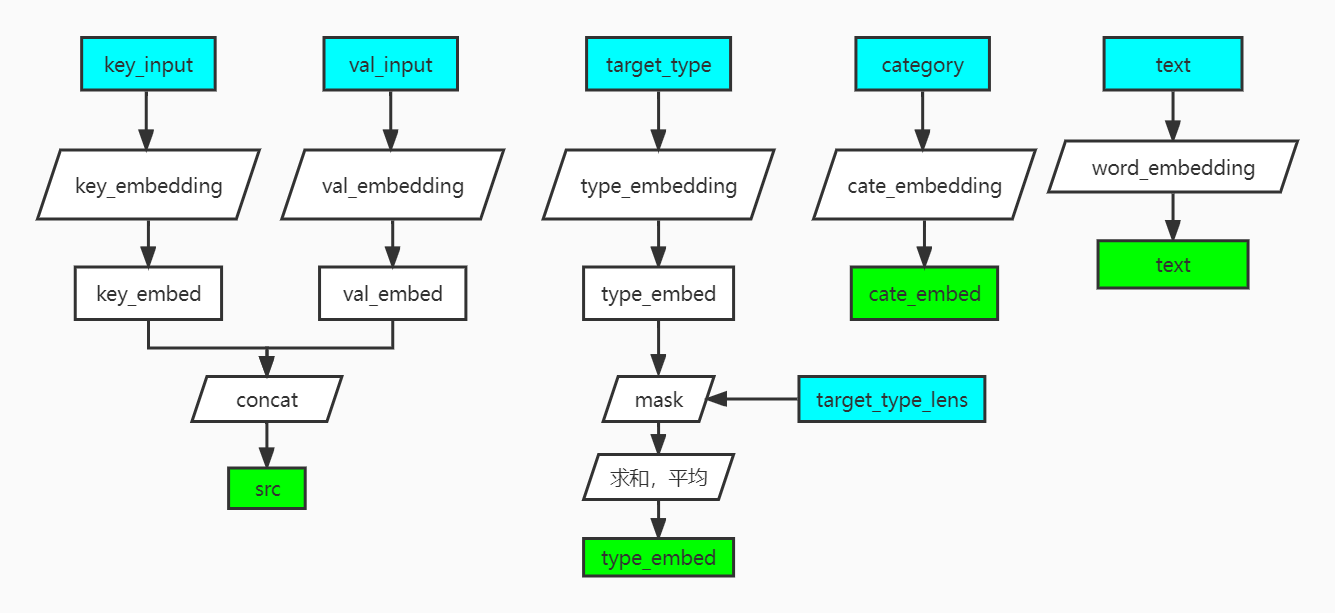
4.feature是order的合集

# 4.Model

**train\_loss=**

**sent\_rec\_loss+group\_rec\_loss+type\_loss+anneal\_sent\_KL+anneal\_plan\_KL**

## 1.embedding



**输入介绍：**

均在dataset的输入内。

**层级介绍：**

1.共5个embed层，输入大小为对应的词表大小，输出为对应类的dim:key\_dim, val\_dim等等。

word\_embedding:句子级，target相关的id的embedding，embedding到word\_dim。

val\_embedding:词级，val相关的id的embedding，embedding到val\_dim。kv对中的v

key\_embedding:词级，key相关的id的embedding，embedding到key\_dim。kv中的v

cate\_embedding:词级，大类相关的id的embedding，embedding到cate\_dim。

type\_embedding:词级，类相关的id的embedding,embedding到type\_dim。

2.concat是连接层，不做运算，将两个结果连接在一起，此处做key\_dim+val\_dim的连接。

3.mask为遮盖层，目的是根据实际的长度:target\_type\_len，将由于规整化的type\_embed的多于部分遮盖（将其实际值化为0），避免影响计算。

**输出介绍：**

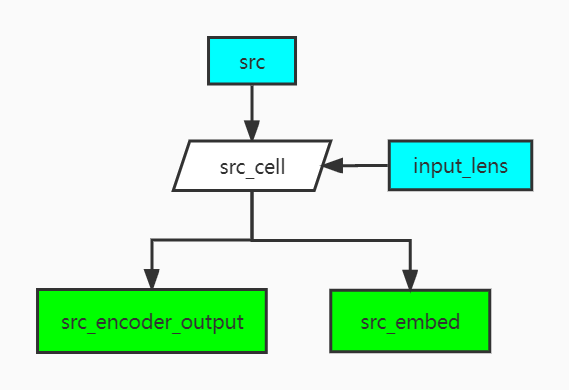
**src**:kv对embedding得到的结果[batch, seq\_kv\_num, key\_dim+val\_dim]

**type\_embed**: target\_type embed得到的结果，每个seg所有的type向量求均值得到最终结果[batch, seg\_num, seg\_type\_num, type\_dim]->[batch, seg\_num, type\_dim]

**cate\_embed**:category embed得到的结果，大类，[batch, cate\_dim]

**text**:对text进行embed，[batch, seq\_word\_num, word\_dim]。

## 2.input\_encode



**输入介绍：**

**src**:出现在 1.embedding 里，是kv对的解码结合得到的。

[batch, seq\_kv\_num, key\_dim+val\_dim]

**input\_lens:**是原始输入，看dataset。[batch,]

**层级介绍：**

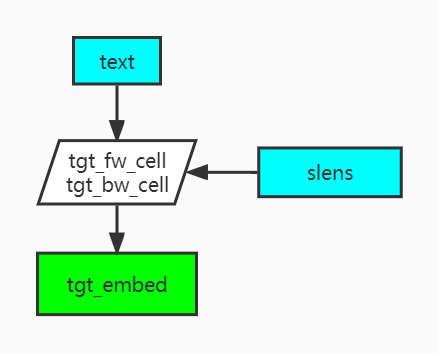
**src\_cell**:encoder层，循环神经网络层，可自定义多少层和输出多少神经元以及是否双向，当双向的时候输出需要concat把结果规整化。

**输出介绍：**

**src\_encoder\_output**:每一时刻的输出，由于是否双向输出有变化，每一时刻代表每一个kv对，其结果由前面的kv对和后面的kv对共同决定，[batch, seq\_kv\_num, encoder\_dim(\*2)].

**src\_embed**:最后一时刻的输出，为整个kv对的结果，即第一个kv对的逆向结果和最后一个kv对的正向结果。 [batch, encoder\_dim(\*2)]

## 3.text\_encode



**输入介绍：**

**text**:出现在 1.embedding 里， 是句子的embed形式。 [batch, seq\_word\_num, word\_dim]

**slens**:输入，在dataset里， [batch,]

**层级介绍：**

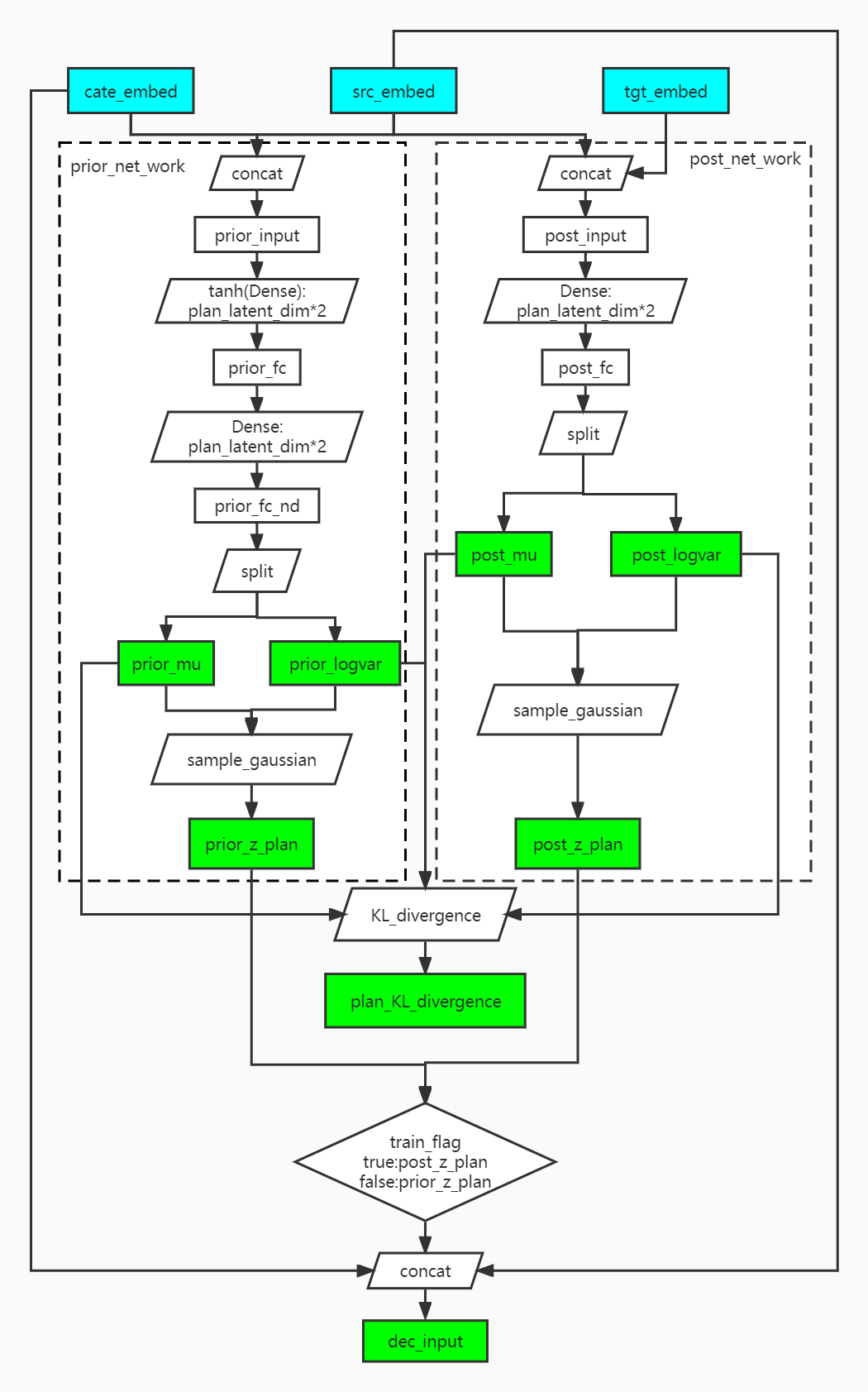
**tgt\_fw\_cell/tgt\_bw\_cell**: 双向循环神经网络，可自定义多少层和多少神经元。输出需要concat得到结果。

**输出介绍：**

**tgt\_embed**:句子embed经历循环神经网络编码得到的结果。

[batch, text\_post\_encoder\_dim\*2]

## 4.top\_level



**输入介绍：**

**cate\_embed**:出现在 1.embedding 里，大类的embed，[batch, cate\_dim]

**src\_embed**:出现在 2.input\_encode 里， 最后一时刻的输出，为整个kv对的结果，即第一个kv对的逆向结果和最后一个kv对的正向结果。 [batch, encoder\_dim(\*2)]

**tgt\_embed**:出现在 3.text\_encode 里， 句子embed经历循环神经网络编码得到的结果。

[batch, text\_post\_encoder\_dim\*2]

**层级介绍：**

**concat**:用于连接，针对最后一维

**dense**:全连接层，将最后一维转换成指定dim，此处dense输出均为plan\_latent\_dim\*2

**split**:这里均分，dense的输出一半是均值，一半是方差

**sample\_gaussian**:重采样，针对得到的均值和方差重采样随机数

**KL\_divergence**:根据两个随机变量的均值和方差计算对应的KL损失

**train\_flag:**判断是否训练

**输出介绍:**

**prior\_mu/prior\_logvar/prior\_z\_plan**:为通过cate\_embed和src\_embed经过全连接得到的均值，方差和通过重采样得到的随机变量，包含有句子大类和所有kv对的信息。[batch, plan\_latent\_dim]。

**post\_mu/post\_logvar/post\_z\_plan**: 为通过cate\_embed和src\_embed和tgt\_embed经过全连接得到的均值，方差和通过重采样得到的随机变量，包含有句子大类和所有kv对和句子本身的信息。[batch, plan\_latent\_dim]。

**dec\_input**:训练时，为cate\_embed,src\_embed,post\_z\_plan的连接，

否则为cate\_embed,src\_embed,prior\_z\_plan。训练时，包含大类的信息，所有kv对的信息，对应的含句子大类和kv对以及句子本身的信息的随机变量（变分思想）；不训练时，包含大类的信息，所有kv对的信息，对应的含句子大类和kv对信息的随机变量。[batch, cate\_dim+encoder\_dim(\*2)+plan\_latent\_dim]

**plan\_KL\_divergence**:根据大类和句子kv对生成的随机变量和再加上句子生成的随机变量之间的KL损失。

## 5.sentence\_level

### 1.group\_sent 循环层构建

**group\_decoder**: 输出 group\_decoder\_dim

**group\_encoder**: 输出group\_encoder\_dim

**latent\_decoder**: 输出latent\_decoder\_dim

**decoder**: 输出 decoder\_dim

### 2.sent\_post\_encoder seg内循环层构建

**sent\_fw\_cell/sent\_bw\_cell**: 输出post\_encoder\_dim

### 3.parameters 全连接层构建

**group\_init\_state\_fc**:输出group\_decoder\_dim

**plan\_init\_state\_fc**:输出 latent\_decoder\_dim

**prior\_fc\_layer**:输出sent\_latent\_dim\*2

**post\_fc\_layer**:输出sent\_latent\_dim\*2

**group\_fc\_1**: 输出 encoder\_dim\*2

**group\_fc\_2**: 输出 1

**type\_fc\_1**: 输出 type\_dim

**type\_fc\_2**: 输出 type\_vocab\_size

**bow\_fc\_1**: 输出 bow\_hidden\_dim

**bow\_fc\_2**: 输出 tgt\_vocab\_size

**projection**: 输出tgt\_vocab\_size

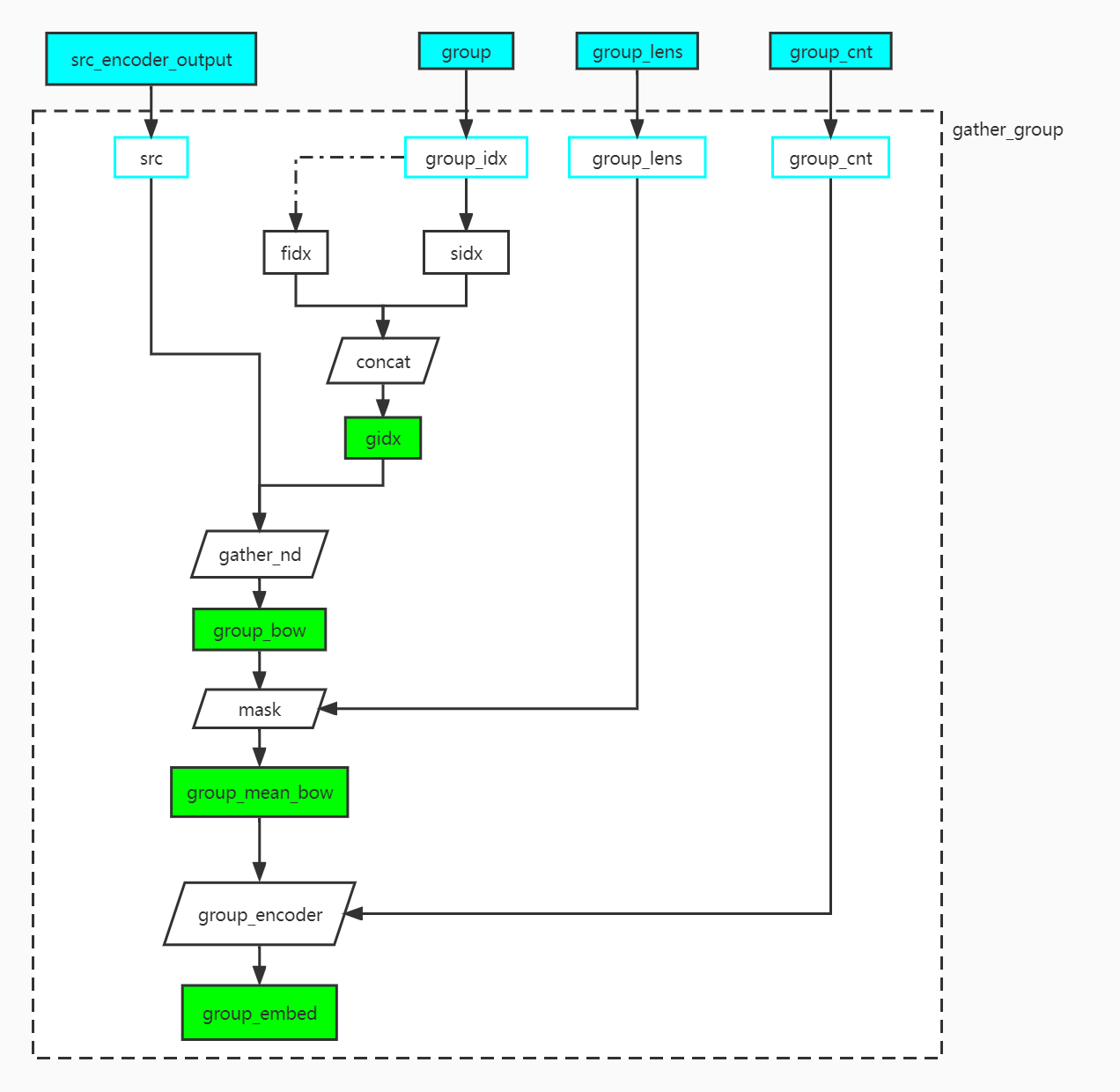
**stop\_clf**: 输出 1

**init\_gbow**: 大小为 [1, encoder\_dim(\*2)]的变量

### 4.train 训练

train针对每一个seg进行训练。对每一个seg做处理，下述前面加有train前缀的为对每个seg处理的介绍。

#### 1.group\_encode



**输入介绍：**

**src\_encoder\_output**:出现在 2.input\_encode 处，每一时刻的输出，由于是否双向输出有变化，每一时刻代表每一个kv对，其结果由前面的kv对和后面的kv对共同决定，[batch, seq\_kv\_num, encoder\_dim(\*2)].

**group**:原始输入，记录了每个seg都有哪些kv对，用kv对的编号进行记录，kv对的编号取决于该句子有多少个kv对，即key\_input和val\_input。[batch, seg\_num, seg\_kv\_num]

**group\_lens**: 原始输入，记录了每个seg的kv对的数量。[batch, seg\_num]

**group\_cnt**: 原始输入，记录了每句话多少个seg。 [batch,]

**层级介绍：**

**gather\_group**: 函数

**蓝框所圈参数**：为函数gather\_group的输入参数，对应输入由图可视。

**concat**:把fidx和sidx的最后一维连接，构成一个序号

**gather\_nd**:依据gidx选取src中对应id的值构成新tensor

**mask**: 根据group\_lens将group\_bow中多余的参数清0

**group\_encoder**:为 5.1 group\_sent 构建的循环层，

**中间变量介绍：**

**fidx**:由group\_idx的形状决定，每个值是对应batch的编号 [batch, seg\_num, seg\_kv\_num, 1]

**sidx**:由group\_idx改变形状得到，每个值对应group的值，[batch, seg\_num, seg\_kv\_num, 1]

**输出介绍：**

**gidx**:由fidx,sidx连接得到，每个值对应为(batch, seq\_kv\_id)的含义，即序号，序号取决于对应句子kv对的编号，记录每个句子每个seg对应的kv对的序号。[batch, seg\_num, seg\_kv\_num, 2]。

**group\_bow**:对应句子对应seg的对应kv对的编码形式。一个kv对对应一个编码，[batch, seg\_num, seg\_kv\_num, encoder\_dim(\*2)]。

**group\_mean\_bow**:对应句子对应seg的整体kv对的编码形式。这里将所有kv对的编码向量求均值得到，一个seg对应一个编码[batch, seg\_num, encoder\_dim(\*2)]

**group\_embed**:输入的time\_step为每句话的seg，值是每句话对应seg的整体kv对的编码形式，这里将所有kv对的编码向量求均值得到，经过循环神经网络得到最后一个时间步的输出。[batch, group\_decoder\_dim]。

#### 2.train\_cond

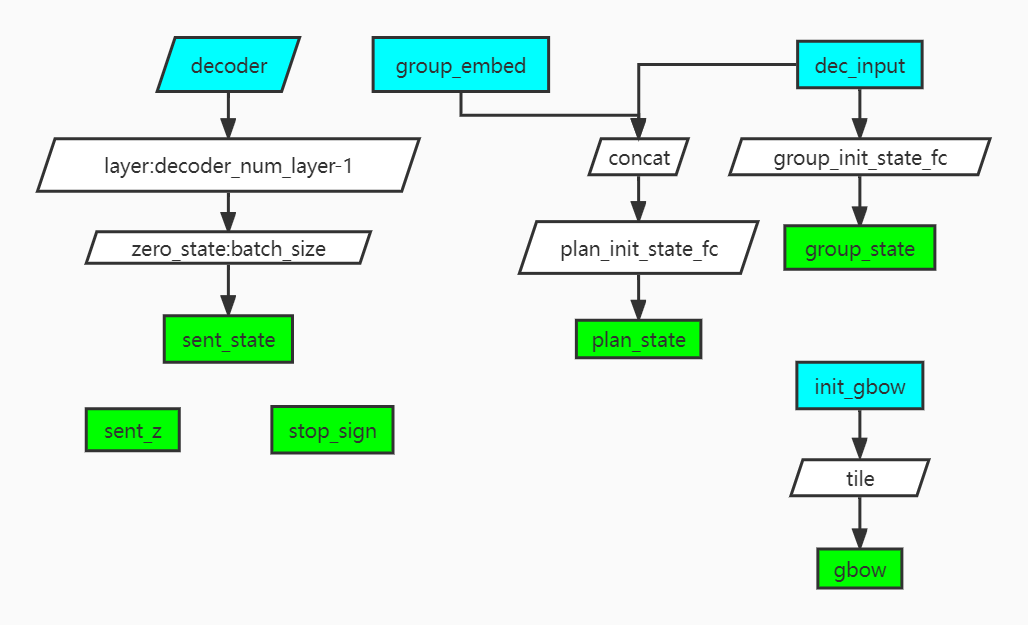
结束循环时的条件：

i: 循环到第i个seg

若 i < seg\_num

这里的seg\_num是由原始输入的output\_lens得到，记录的seg\_num大小是所有句子中最大的seg数目。

#### 3.train\_init



**输入介绍:**

**decoder**: 5.1 group\_sent构建的循环层，未经过使用，

**group\_embed**: 5.4.1 group\_encode得到的输出，输入的time\_step为每句话的seg，值是每句话对应seg的整体kv对的编码形式，这里将所有kv对的编码向量求均值得到，经过循环神经网络得到最后一个时间步的输出。[batch, group\_decoder\_dim]。

**dec\_input**: 4.top\_level中的输出，训练时，为cate\_embed,src\_embed,post\_z\_plan的连接，

否则为cate\_embed,src\_embed,prior\_z\_plan。训练时，包含大类的信息，所有kv对的信息，对应的含句子大类和kv对以及句子本身的信息的随机变量（变分思想）；不训练时，包含大类的信息，所有kv对的信息，对应的含句子大类和kv对信息的随机变量。[batch, cate\_dim+encoder\_dim(\*2)+plan\_latent\_dim]

**init\_gbow**:5.3 parameters得到的随机生成的变量 大小为 [1, encoder\_dim(\*2)]的变量

**层级介绍：**

**layer&&zero\_state**：decoder记录了每一层的状态，取最后一层状态，然后用zero\_state初始化到batch\_size的大小。

**concat**:连接

**plan\_init\_state\_fc**:5.3 sent\_post\_encoder中出现， 输出latent\_decoder\_dim，用于初始化plan\_state

**group\_init\_state\_fc**:5.3 sent\_post\_encoder 中出现， 输出group\_decoder\_dim，用于初始化group\_state。

**tile**:将init\_gbow复制扩充至batch大小，[batch, encoder\_dim(\*2)]。

**输出介绍：**

**sent\_state**:初始化用于送入循环体的sent\_state，[batch, decoder\_dim]

**sent\_z**:初始化用于送入循环体的sent\_z，初始化为0，[batch, sent\_latent\_dim]

**stop\_sign**:初始化用于送入循环体的stop\_sign，初始化为[seg\_num,None]

**plan\_state**:初始化用于送入循环体的plan\_state，借助dec\_input和group\_embed初始化，包含有句子整体kv对的信息（group\_encoder和src\_cell），大类信息，以及对应的随机变量，训练时除src\_cell得到的kv对信息，大类信息，还有对应句子得到的编码信息整合得到随机变量。[batch, latent\_decoder\_dim]

**group\_state**:初始化用于送入循环体的plan\_state，借助dec\_input进行初始化，包含大类信息，句子整体kv对的信息，训练时还含有对应句子编码得到的信息。[batch, group\_decoder\_dim]

**gbow**:由init\_gbow得来并扩充至batch大小，[batch, encoder\_dim(\*2)]

#### 4.train\_body

**gbow**:作为输入时为前一时刻的seg对应的整体kv对的编码形式，更新后为当前seg对应的整体kv对的编码形式。[batch, encoder\_dim(\*2)]

**stop\_sign**:用于记录第i个seg是否停止的标准，最终为[seg\_num, batch]

**group\_rec\_loss**:评估对应句子对应seg中，是否出现对应kv对的损失。group\_label为第i个seg中，有哪些kv对，group\_logit为预测此的结果。float, 累加上所有seg的损失，最终为[batch,]

**group\_state**: 为group\_decoder的隐藏状态，group\_decoder为 5.1 group\_sent 中定义的循环层，作为输入是前一时刻group\_decoder的状态，输出是当前时刻group\_decoder的状态。[batch, group\_decoder\_dim]

**sent\_state**: 为basic\_decoder解码后的状态，即解码出来当前seg句子后的decoder状态。[batch, decoder\_dim]

**plan\_state**:为latent\_decoder上一时刻的状态，主要用于seg传递decoder状态，[batch, latent\_decoder\_dim]

**KL\_loss:** 即通过kv对和上一句子产生的随机变量和实际句子的信息的随机变量差异，这些随机变量用于产生句子。主要在于是否包含当前seg句子的编码（sent\_embed）和type编码(sent\_type\_embed) KL散度[batch, ]- mask后求和为1个值。

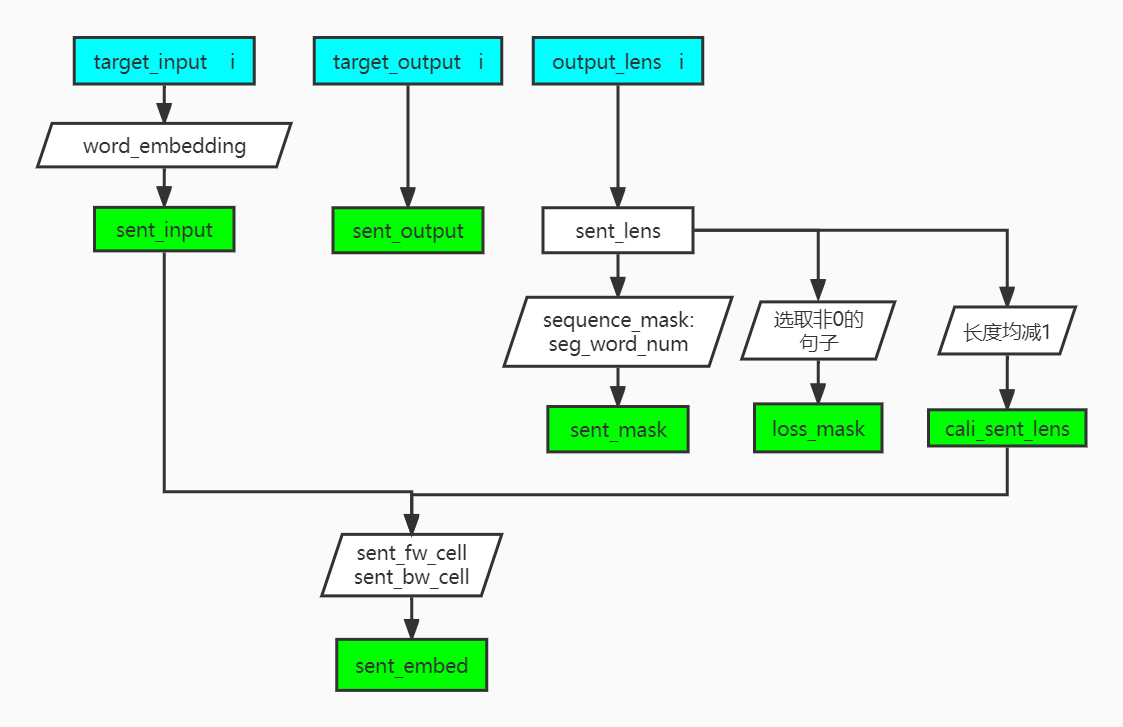
**sent\_z**:经历重采样得到的随机变量信息，包含有seg句子编码，整体kv对编码，整体type编码的信息和之前状态的信息。[batch, sent\_latent\_dim]

**type\_loss**: 类型损失，判断当前seg有哪些type。 一个值

**sent\_rec\_loss**:是比较生成的句子和实际的句子差异的损失，为单独的一个值

**bow\_loss**：判断该句子该seg有哪些词的损失，不要求顺序。

##### 1.sent



**输入介绍：**

**target\_input:**原始输入，每个句子每个seg的句子，于前面加上开始符，表示为词表id，[batch, seg\_num, seg\_i\_word\_num]。

**target\_output**:原始输入，每个句子每个seg的句子，于后面加上结束符，表示为词表id，[batch, seg\_num, seg\_i\_word\_num]。

**output\_lens:**原始输入，每个句子中每个seg的长度，[batch, seg\_num, seg\_word\_num]

**层级介绍：**

**word\_embedding**: 出现在 1. embedding里，target相关的id的embedding，embedding到word\_dim。

**sequence\_mask:**方法，用于将sent\_lens的长度表示，改变成对应seg\_word\_num长的mask tensor。mask长于sent\_lens中数值的word

选取非0的句子：将sent\_lens长度为0的mask掉。

长度均减1：将sent\_lens的值统一减1，0不用变。

**sent\_fw\_cell/sent\_bw\_cell**:出现在 5.2 sent\_post\_encoder 中，建立的双向循环层

**输出介绍：**

**sent\_input:**每个句子第i个seg句子的id形式，[batch, seg\_i\_word\_num]

**sent\_output**:每个句子第i个seg句子的id形式，[batch, seg\_i\_word\_num]

**sent\_mask**:每个句子第i个seg句子的mask，mask掉对应句子的padding的词，

[batch, seg\_i\_word\_num]

**loss\_mask:**对应seg的每个句子的mask，mask掉长度为0的句子。[batch,]

**cali\_sent\_lens**:是sent\_lens所有值减1的结果，为排除作为输入的target\_input的开始符的长度。[batch,]

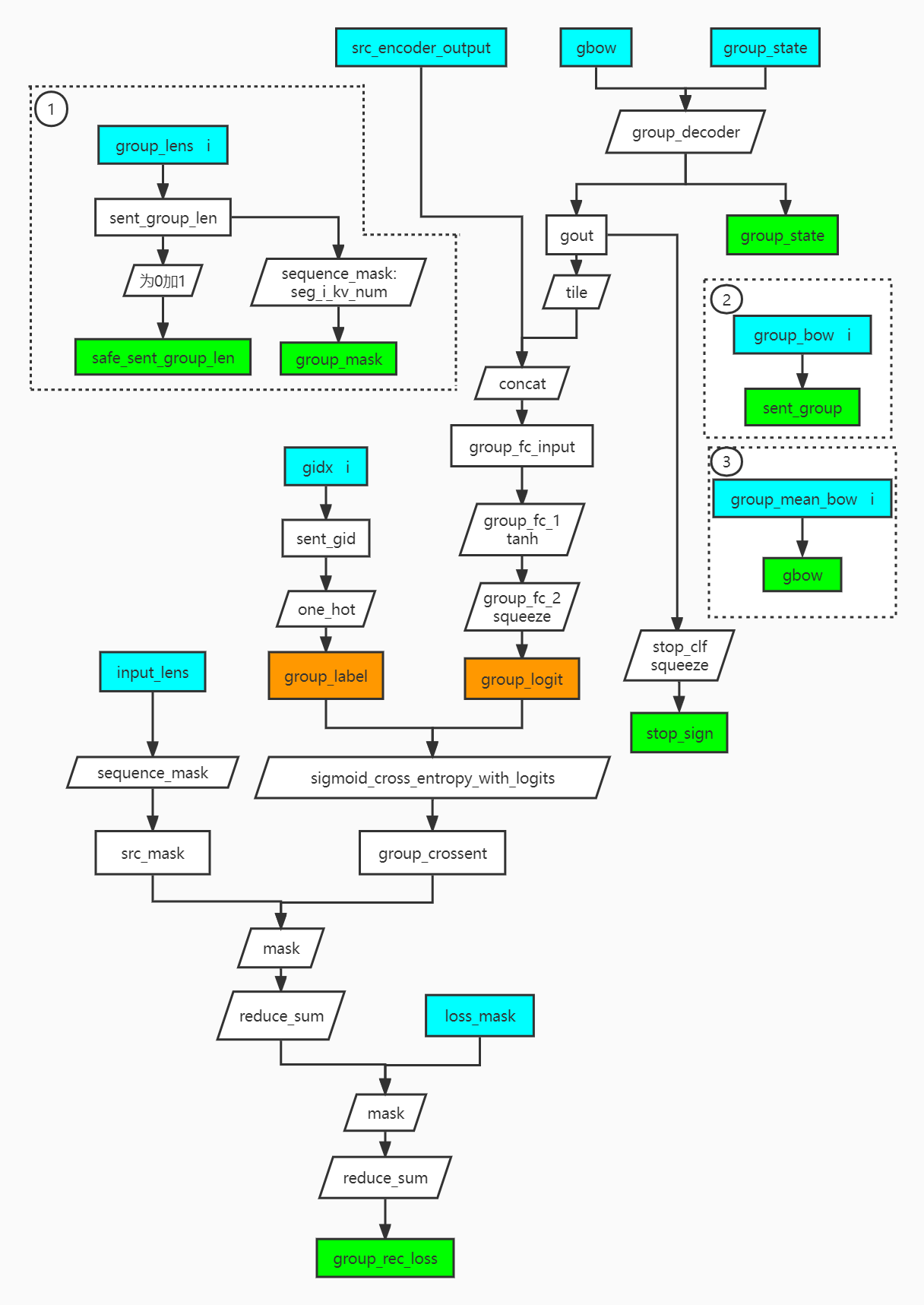
**sent\_embed**:对应句子第i个seg句子经历循环层得到的最终状态，表示该seg句子的编码。

[batch, sent\_post\_encoder\_dim\*2]

##### 2.group

可以看出，下述结构图存在过多的图的割裂，不是一个整体，但是代码此处构成均为group域的内容，因此这样画图。

为方便理解把group图分为4部分，part1, part2, part3,以及主干部分(decode\_group)。



在叙述网络前，先对此处网络出现的train\_body参数，进行总结：

**gbow**:在此图作为输入，并更新，作为输入时为前一时刻的seg对应的整体kv对的编码形式，更新后为当前seg对应的整体kv对的编码形式。[batch, encoder\_dim(\*2)]

**stop\_sign**:此图只用作更新，用于记录第i个seg是否停止的标准，最终为[seg\_num, batch]

**group\_rec\_loss**:评估对应句子对应seg中，是否出现对应kv对的损失。group\_label为第i个seg中，有哪些kv对，group\_logit为预测此的结果。float, 累加上所有seg的损失，最终为[batch,]

**group\_state**: 为group\_decoder的隐藏状态，group\_decoder为 5.1 group\_sent 中定义的循环层，这里作为输入是前一时刻group\_decoder的状态，输出是当前时刻group\_decoder的状态。[batch, group\_decoder\_dim]

###### 1.part1

**输入介绍：**

**group\_lens**:为原始输入，记录了每个seg的kv对的数量,[batch, seg\_num]

**层级介绍：**

**为0加1**：为了使safe\_sent\_group\_len作为除数不含0

**sequence\_mask:**将输入的值变为长seg\_i\_kv\_num的tensor的mask

**输出介绍：**

**safe\_sent\_group\_len**:是group\_lens的第i个seg的值，记录了每个句子第i个seg的kv对数量，如果kv对数为0，则加1.[batch,]

**group\_mask**:mask掉对应seg多余的kv对位置，这些地方没有实际kv对，但由于长度是所有seg中最多kv对的数量，所以需要mask。[batch, seg\_i\_kv\_num]

###### 2.part2

**输入介绍：**

**group\_bow**: 出现在5.4.1 group\_encode部分，对应句子对应seg的对应kv对的编码形式。一个kv对对应一个编码，[batch, seg\_num, seg\_kv\_num, encoder\_dim(\*2)]。

**输出介绍：**

**sent\_group**: 为对应句子第i个seg的对应kv对的编码形式，一个kv对对应一个编码 [batch, seg\_i\_kv\_num, encoder\_dim(\*2)]

###### 3.part3

**输入介绍：**

**group\_mean\_bow**: 出现在5.4.1 group\_encode部分，对应句子对应seg的整体kv对的编码形式。这里将所有kv对的编码向量求均值得到，一个seg对应一个编码[batch, seg\_num, encoder\_dim(\*2)]。

**输出介绍：**

**gbow**: 为train\_body的关键单元，为当前seg对应的整体kv对的编码形式。[batch, encoder\_dim(\*2)]

###### 4.decode\_group

**输入介绍：**

**src\_encoder\_output**:出现在2 input\_encode中，针对整个句子的kv对，每一时刻的输出，由于是否双向输出有变化，每一时刻代表每一个kv对，其结果由前面的kv对和后面的kv对共同决定，[batch, seq\_kv\_num, encoder\_dim(\*2)].

**gbow**:出现在5.4.4.2.3 part3里，这里表示前一个seg对应的整体kv对的编码形式，[batch, encoder\_dim(\*2)]

**group\_state**: 为group\_decoder的隐藏状态，group\_decoder为 5.1 group\_sent 中定义的循环层，作为输入是前一时刻group\_decoder的状态，输出是当前时刻group\_decoder的状态。[batch, group\_decoder\_dim]

**gidx**:出现在5.4.1 group\_encode 中， 每个值对应为(batch, seq\_kv\_id)的含义，即序号，序号取决于对应句子kv对的编号，记录每个句子每个seg对应的kv对的序号。[batch, seg\_num, seg\_i\_kv\_num, 2]。

**input\_lens**:原始输入，是整个句子中kv对的数量 [batch,]

**loss\_mask**:出现在5.4.4.1 sent 部分，为对应seg的每个句子的mask，mask掉长度为0的句子。[batch,]

**中间变量介绍：**

**gout**:包含有seg中所有kv对信息解码得到的输出，由group\_decoder得到，[batch, group\_decoder\_dim]

**group\_fc\_input**:将gout进行复制扩展(tile)到[batch, seq\_kv\_num, group\_decoder\_dim]， 然后同src\_encoder\_output进行连接（concat）。包含有句子所有kv对的信息，当前seg的整体kv对的信息，以及前面seg的时序信息。[batch, seq\_kv\_num, group\_decoder\_dim+encoder\_dim(\*2)]

**group\_logit**:是group\_fc\_input经过两层全连接层得到的，包含有句子所有kv对的信息，当前seg的整体kv对的信息，以及前面seg的时序信息，对应为第i个seg有哪些kv对

**sent\_gid**: 为第i个seg所有kv对的序号，每个值对应为(batch, seq\_kv\_id)的含义，即序号，序号取决于对应句子kv对的编号，[batch, seg\_i\_kv\_num, 2]。

**group\_label**: 为第i个seg有哪些kv对，[batch, seq\_kv\_num]。

**group\_crossent**:group\_label和group\_logit原始的交叉熵，[batch, seq\_kv\_num]

**src\_mask**: input\_lens的mask的tensor, [batch, seq\_kv\_num]

**层级介绍：**

**group\_decoder**:为5.1 group\_sent中构建的循环层。输入是指定句子对应seg中整体kv对的编码信息以及decoder上一次的状态，这里由于是针对单独一个seg做运算，gout没有时间步的维度。

**group\_fc\_1**：为5.3 parameters 中构建的全连接层，输出为encoder\_dim\*2,在后面加上tanh激活层。此时为[batch, seq\_kv\_num, encoder\_dim\*2]

**group\_fc\_2**：为5.3parameters 中构建的全连接层，输出为1，在后面加上squeeze，去除长只有1的维度。 此时为[batch, seq\_kv\_num]

**one\_hot**:把sent\_gid的关于kv对的编号变为one\_hot格式，长度为seq\_kv\_num。[batch, seg\_i\_kv\_num, seq\_kv\_num]，之后根据group\_mask(5.4.4.2.1part1)mask后，执行了累加，表示为第i个seg有哪些kv对，为[batch, seq\_kv\_num]。

**sigmoid\_cross\_entropy\_with\_logits**:求对应的交叉熵，即group\_label和group\_logit的交叉熵。输出为[batch, seq\_kv\_num]

**sequence\_mask**:将input\_lens转换成mask的tensor,[batch, seq\_kv\_num]

**stop\_clf:**为5.3parameters中构建的全连接层，输出为1，后接squeeze去除长度为1的维度。

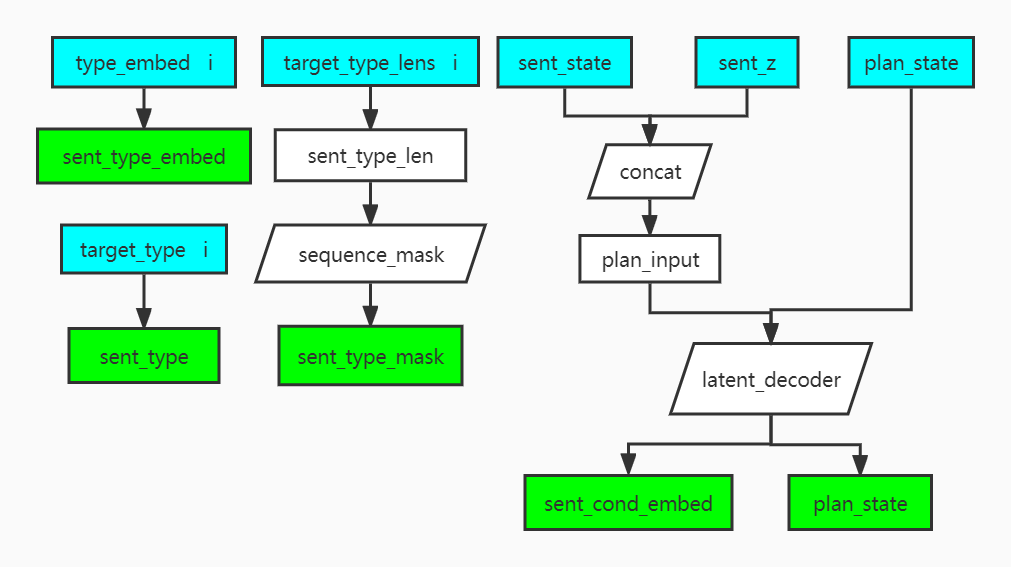
**输出介绍：**

**group\_state**:为当前group\_decoder的状态，包含的信息是seg中所有kv对解码得到的信息。[batch, group\_decoder\_dim]

**stop\_sign**:停止符，记录第i个seg是否停止的标准，最终为[seg\_num, batch]

**group\_rec\_loss**：此处的实现是由group\_crossent和src\_mask相乘mask后，再求和得到[batch, ], 之后再根据loss\_mask对应mask后求和得到一个值，该值为所有分损失的和。评估对应句子对应seg中，是否出现对应kv对的损失。[batch,]

##### 3.type and plan



**关键变量：**

**sent\_state**:[batch, decoder\_dim] 为basic\_decoder解码后的状态，即解码出来当前seg句子后的decoder状态。

**sent\_z**:[batch, sent\_latent\_dim]，经历重采样得到的随机变量信息，包含有seg句子编码，整体kv对编码，整体type编码的信息和之前状态的信息。

**plan\_state**:为latent\_decoder上一时刻的状态，主要用于seg传递decoder状态，[batch, latent\_decoder\_dim]

**输入介绍：**

**type\_embed**:出现在2.input\_encode中，每个seg所有的type向量求均值得到最终结果 [batch, seg\_num, type\_dim]

**target\_type**:原始输入，记录每个seg都描述了什么类。为id形式。[batch, seg\_num, seg\_\_i\_type\_num]

**target\_type\_lens**:原始输入，记录每个seg描述的类的数量，[batch, seg\_num]

**sent\_state**: [batch, decoder\_dim]，为basic\_decoder解码后的状态，即解码出来当前seg句子后的decoder状态。

**sent\_z**: [batch, sent\_latent\_dim]，上一状态，经历重采样得到的随机变量信息，包含有seg句子编码，整体kv对编码，整体type编码的信息和之前状态的信息。

**plan\_state:** latent\_decoder上一时刻的状态，主要用于seg传递decoder状态，[batch, latent\_decoder\_dim]

**层级介绍：**

**sequence\_mask**:构建对应seg描述类的数量的mask，mask到seg\_i\_type\_num。输入是[batch,]，来自于target\_type\_lens的第i个seg的值。

**concat**:连接

**latent\_decoder**:出现在5.1group\_sent里，输出latent\_decoder\_dim

**输出介绍：**

**sent\_type\_embed**:第i个seg所有的type向量求均值得到最终结果 [batch, type\_dim]

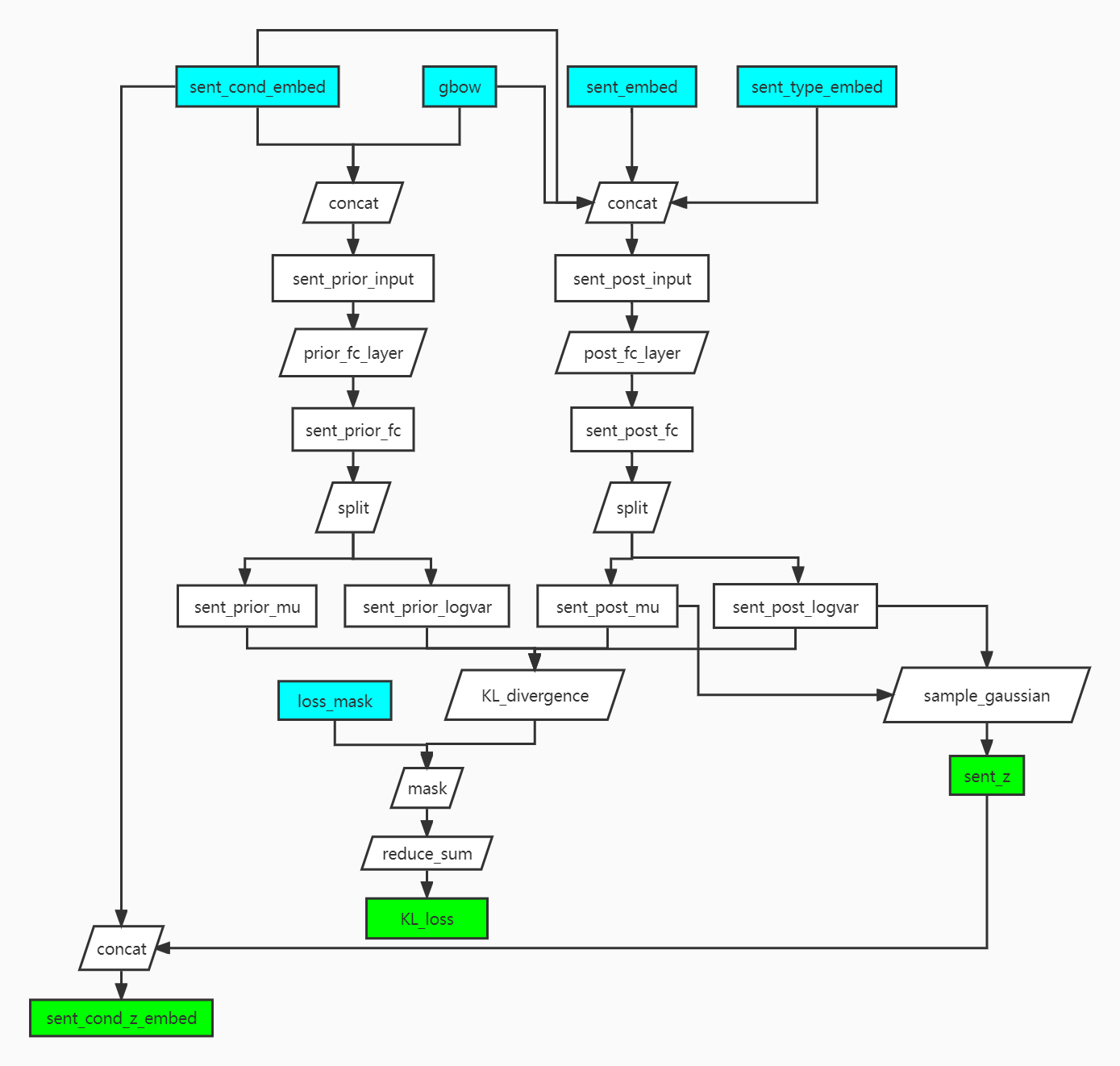
**sent\_type:**第i个seg描述了什么类 [batch, seg\_\_i\_type\_num]

**sent\_type\_mask**:第i个seg描述的类的数量的mask，[batch, seg\_i\_type\_num]

**sent\_cond\_embed**:来自sent\_state和sent\_z的信息，加上latent\_decoder之前的状态解码得到的输出[batch, latent\_decoder\_dim]

**plan\_state**：latent\_decoder当前时刻的状态，主要用于seg传递decoder状态，[batch, latent\_decoder\_dim]

##### 4.sent\_net\_work



**关键元素：**

**KL\_loss:** 上一状态的sent信息a(sent\_cond\_embed)，当前seg对应的整体kv对的编码b(gbow)，对应句子第i个seg句子经历循环层得到的最终状态，表示该seg句子的编码c(sent\_embed)，第i个seg所有的type向量求均值得到最终结果d(sent\_type\_embed)。a和b构造的随机变量和abcd构造的随机变量的损失。

即通过kv对和上一句子产生的随机变量和实际句子的信息的随机变量差异，这些随机变量用于产生句子。

这里针对ab/abcd concat后再经历全连接层得到指定均值方差，求KL散度，再通过loss\_mask清0长度为0的句子的损失，求和加到KL\_loss上，最终为一个值。

KL散度[batch, ]- mask后求和为1个值。

**sent\_z**: seg句子信息，整体kv对编码，以及整体type编码和上一状态的sent信息经历全连接层得到均值方差求的随机变量的隐含变量。 [batch, sent\_latent\_shape]

**输入介绍：**

**sent\_cond\_embed**: (5.4.4.3 type\_and\_plan) 来自sent\_state和sent\_z的信息，加上latent\_decoder之前的状态解码得到的输出[batch, latent\_decoder\_dim]

**gbow**: (5.4.4 train\_body)为train\_body的重要元素，此时为当前seg对应的整体kv对的编码形式。[batch, encoder\_dim(\*2)]。

**sent\_embed**: (5.4.4.1 sent) 对应句子第i个seg句子经历循环层得到的最终状态，表示该seg句子的编码。[batch, sent\_post\_encoder\_dim\*2]

**sent\_type\_embed**: (5.4.4.3 type\_and\_plan) 第i个seg所有的type向量求均值得到最终结果 [batch, type\_dim]

**loss\_mask**: (5.4.4.1 sent) 对应seg的每个句子的mask，mask掉长度为0的句子。[batch,]

**层级介绍：**

**prior\_fc\_layer**: (5.3 parameters) 全连接层 输出sent\_latent\_dim\*2

**post\_fc\_layer**: (5.3 parameters) 全连接层 输出sent\_latent\_dim\*2

**KL\_divergence**: 根据两个随机变量的均值方差共4个参数得到随机变量相似度KL损失

**sample\_gaussian**: 根据一个随机变量的均值和方差重采样得到对应随机变量。

**输出介绍：**

**KL\_loss:** 上一状态的sent信息a(sent\_cond\_embed)，当前seg对应的整体kv对的编码b(gbow)，对应句子第i个seg句子经历循环层得到的最终状态，表示该seg句子的编码c(sent\_embed)，第i个seg所有的type向量求均值得到最终结果d(sent\_type\_embed)。a和b构造的随机变量和abcd构造的随机变量的损失。

即通过kv对和上一句子产生的随机变量和实际句子的信息的随机变量差异，这些随机变量用于产生句子。

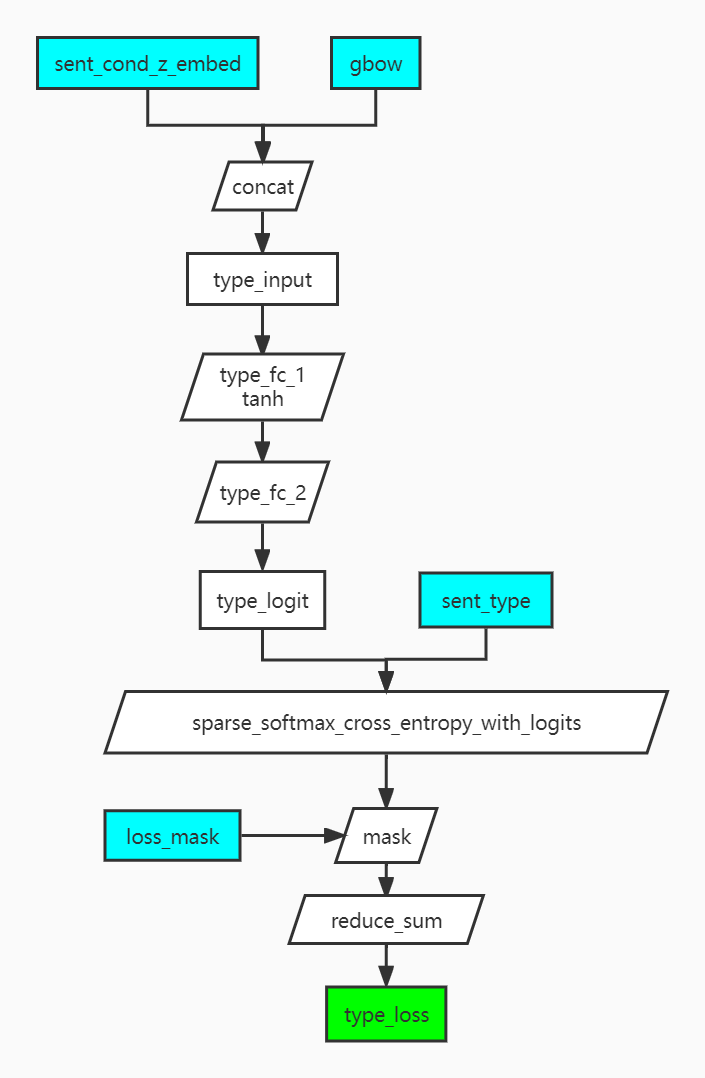
这里针对ab/abcd concat后再经历全连接层得到指定均值方差，求KL散度，再通过loss\_mask清0长度为0的句子的损失，求和加到KL\_loss上，最终为一个值。

KL散度[batch, ]- mask后求和为1个值。

**sent\_z**: seg句子信息，整体kv对编码，以及整体type编码和上一状态的sent信息经历全连接层得到均值方差求的随机变量的隐含变量。 [batch, sent\_latent\_shape]

**sent\_cond\_z\_embed**:由上一状态的sent信息和当前状态产生的sent信息得到,包含有seg句子信息，整体kv对编码，type编码，上一时刻的sent信息。 [batch, latent\_decoder\_dim+sent\_latent\_dim]

##### 5.type\_loss



**关键元素：**

**type\_loss**: 类型损失，判断当前seg有哪些type。 一个值

**输入介绍：**

**sent\_cond\_z\_embed**:sent\_state和sent\_z decode得到的信息，连接上当前重采样生成的sent\_z的信息，包含之前状态的信息，seg句子，整体kv对编码和整体type编码的信息。[batch, latent\_decoder\_dim+sent\_latent\_dim]

**gbow:** (5.4.4 train\_body) 当前seg对应的整体kv对的编码形式。[batch, encoder\_dim(\*2)]

**sent\_type:** (5.4.4.3 type\_and\_plan) 第i个seg描述了什么类 [batch, seg\_\_i\_type\_num]

**层级介绍：**

**type\_fc\_1**: (5.3 parameters) 输出type\_dim 后面跟tanh激活层

**type\_fc\_2**: (5.3 parameters) 输出type\_vocab\_size.

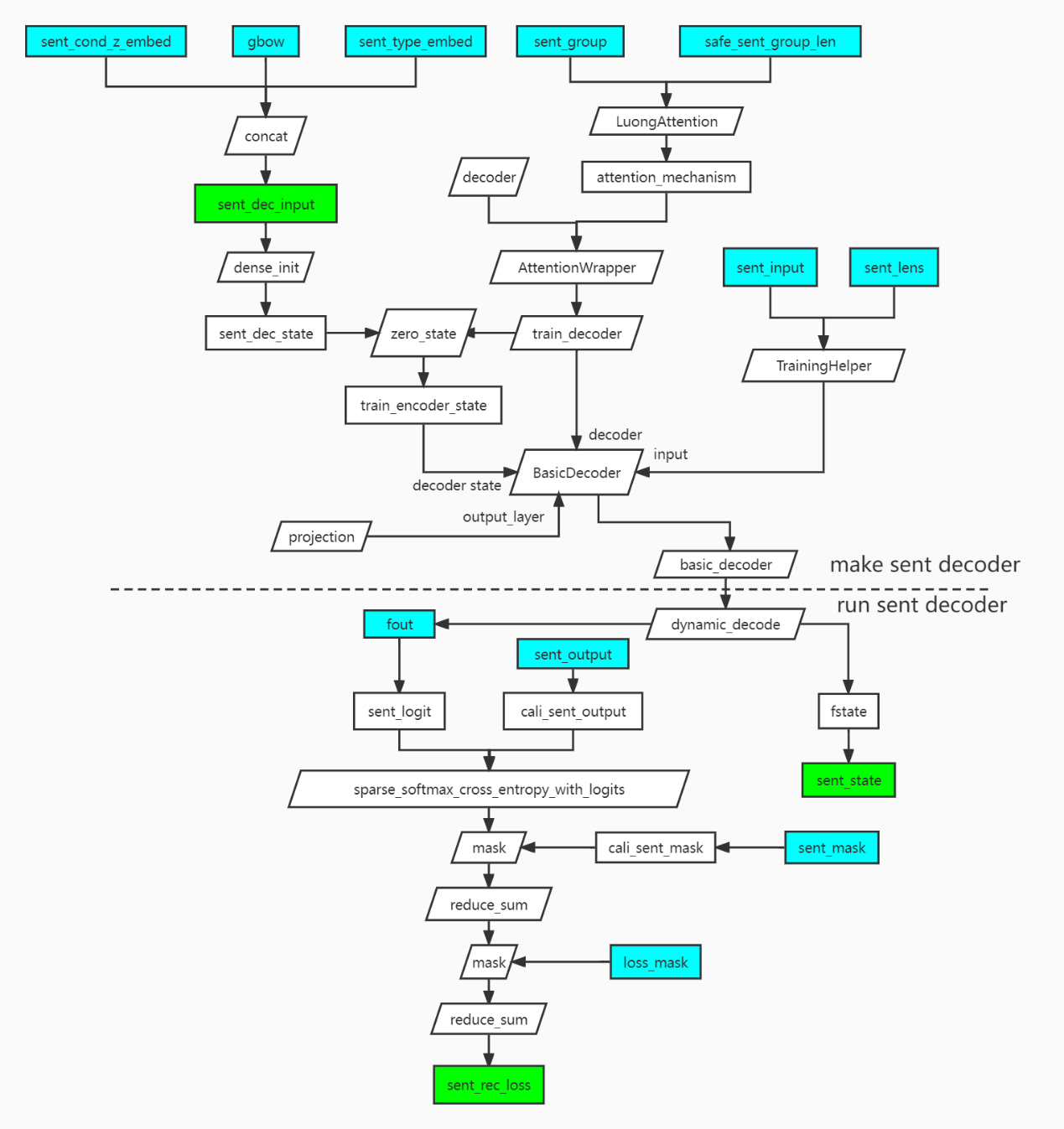
**sparse\_softmax\_cross\_entropy\_with\_logits**:求交叉熵，输入sent\_type[batch, seg\_\_i\_type\_num], type\_logit[batch, seg\_i\_type\_num, type\_vocab\_size]

**输出介绍：**

**type\_loss**：类型损失 判断当前seg有哪些type，经历交叉熵得到 [batch,]然后通过loss\_mask掉为0值，之后求和得到一个值。

##### 6.sent\_decode

由于sent\_decode过大，分为两部分make\_sent\_decoder和run\_sent\_decoder。



先把整幅图出现的train\_body参数进行介绍：

**gbow**: (5.4.4.2.3 part3) 为当前seg对应的整体kv对的编码形式。[batch, encoder\_dim(\*2)]

**sent\_state**: 为basic\_decoder解码后的状态，即解码出来当前seg句子后的decoder状态。[batch, decoder\_dim]

**sent\_rec\_loss**:是比较生成的句子和实际的句子差异的损失，为单独的一个值。

###### 1.make sent decoder

**输入介绍：**

**sent\_cond\_z\_embed**: (5.4.4.4 sent net work) 由上一状态的sent信息(sent\_state&sent\_z)和当前状态产生的sent信息得到,包含有seg句子信息，整体kv对编码，type编码，上一时刻的sent信息。 [batch, latent\_decoder\_dim+sent\_latent\_dim]

**gbow:** (5.4.4.2.3 part3) 为当前seg对应的整体kv对的编码形式。[batch, encoder\_dim(\*2)]

**sent\_type\_embed**: (5.4.4.3 type and plan) 第i个seg所有的type向量求均值得到最终结果 [batch, type\_dim]

**sent\_group**: (5.4.4.2.2 part2) 为对应句子第i个seg的对应kv对的编码形式，一个kv对对应一个编码 [batch, seg\_i\_kv\_num, encoder\_dim(\*2)]

**safe\_sent\_group\_len**: (5.4.4.2.1 part1) 是group\_lens的第i个seg的值，记录了每个句子第i个seg的kv对数量，如果kv对数为0，则加1.[batch,]

**sent\_input**: (5.4.4.1 sent) 每个句子第i个seg句子的embed形式，[batch, seg\_i\_word\_num]

**sent\_lens**: (5.4.4.1 sent)为每个句子第i个seg有多长 [batch,]

**层级介绍：**

**dense\_init**: 将sent\_cond\_z\_embed，gbow和sent\_type\_embed连接后得到sent\_dec\_input，通过自动随机生成dense层的参数，使sent\_dec\_input通过dense初始化为sent\_dec\_state。使得sent\_dec\_state初始化含上一状态信息，当前sent重采样信息(sent\_z)，整体kv对信息，type编码信息，使得decoder的状态更接近实际。

**zero\_state**: 初始化train\_decoder的状态，为sent\_dec\_state，得到train\_encoder\_state。

**LuongAttention**:将sent\_group依据safe\_sent\_group\_len进行attention封装得到attention\_mechanism。

**decoder**: (5.1 group\_sent)循环层 输出decoder\_dim

**AttentionWrapper:**为循环层的前面加上attention， 这里将decoder和attention\_mechanism封装成train\_decoder，为一个含attention机制的循环层。

**projection:** (5.3 parameters) 全连接 输出tgt\_vocab\_size.

**TrainingHelper**:封装序列化label，将sent\_input按照sent\_lens的长度输出。

**BasicDecoder**: 封装decoder，子decoder: train\_decoder，输入为train\_decoder中attention机制的输入，sent\_group。初始化状态：train\_decoder\_state，label：TrainingHelper封装，输出全连接层：projection。 最终是由kv对输出对应的描述句子。

**输出介绍：**

**sent\_dec\_input**: 由上一状态的sent信息(sent\_state&sent\_z)和当前状态产生的sent信息得到,包含有seg句子信息，整体kv对编码，type编码，上一时刻的sent信息。以及直接包含的整体kv对编码信息和整体type编码信息。[batch, latent\_decoder\_dim + encoder\_dim +type\_dim]

###### 2. run sent decoder

**输入介绍：**

**fout**: 为basic\_decoder的输出，里面包含了两种结果的表现形式，output:[batch, seg\_i\_word\_num, tgt\_vocab\_size]为选中指定词的概率。 sample\_id[batch, seg\_i\_word\_num]为选中的词的id。

**sent\_mask**: (5.4.4.1 sent) 每个句子第i个seg句子的mask，mask掉对应句子的padding的词，[batch, seg\_i\_word\_num]

**loss\_mask**: (5.4.4.1 sent) 对应seg的每个句子的mask，mask掉长度为0的句子。[batch,]

**sent\_output:** (5.4.4.1 sent) 每个句子第i个seg句子的embed形式，[batch, seg\_i\_word\_num]

**层级介绍**

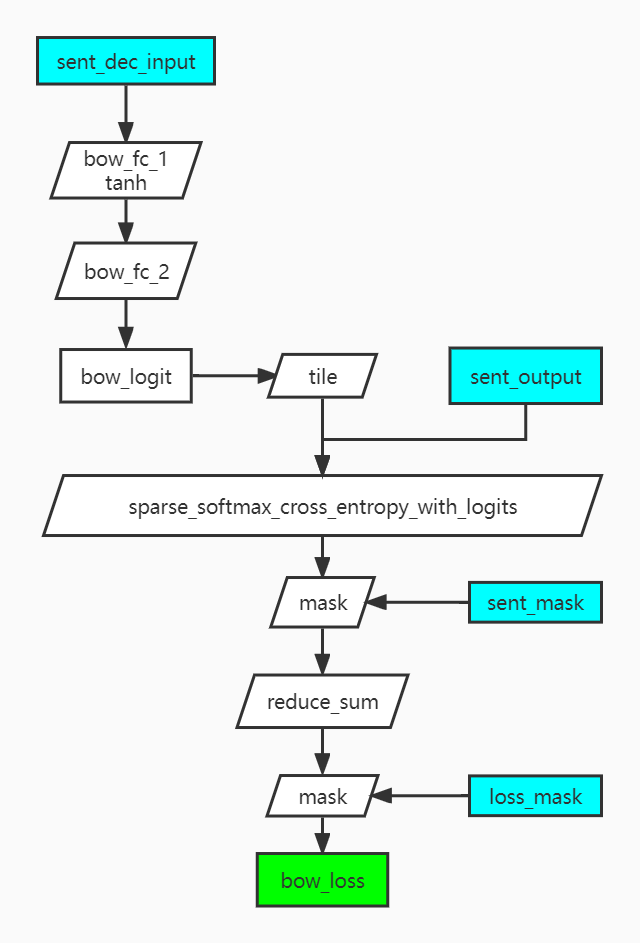
**sparse\_softmax\_cross\_entropy\_with\_logits**:求交叉熵，为sent\_logit(fout的output):[batch, seg\_i\_word\_num, tgt\_vocab\_size]和cali\_sent\_output: [batch, seg\_i\_word\_num]求交叉熵，表示为预计的seg的句子和实际seg的句子的误差，得到[batch, seg\_i\_word\_num]大小的loss。之后先经历cali\_sent\_mask，mask掉为空的词的损失，然后求和得到[batch,]，再根据loss\_mask，mask掉为空的句子的损失，然后求和得到最终的损失。

**输出介绍：**

**sent\_state**: 为basic\_decoder解码后的状态，即解码出来当前seg句子后的decoder状态。[batch, decoder\_dim]

**sent\_rec\_loss**: 是比较生成的句子和实际的句子差异的损失，为单独的一个值。

##### 7.bow\_loss



**输入介绍：**

**sent\_dec\_input**: (5.4.4.6.1 make sent decoder) 由上一状态的sent信息(sent\_state&sent\_z)和当前状态产生的sent信息得到,包含有seg句子信息，整体kv对编码，type编码，上一时刻的sent信息。以及直接包含的整体kv对编码信息和整体type编码信息。[batch, latent\_decoder\_dim + encoder\_dim +type\_dim]

**sent\_output:** (5.4.4.1 sent) 每个句子第i个seg句子的id形式，[batch, seg\_i\_word\_num]

**sent\_mask**: (5.4.4.1 sent) 每个句子第i个seg句子的mask，mask掉对应句子的padding的词，[batch, seg\_i\_word\_num]

**loss\_mask**: (5.4.4.1 sent) 对应seg的每个句子的mask，mask掉长度为0的句子。[batch,]

**层级介绍：**

**bow\_fc\_1**: (5.3 parameters) 输出为 bow\_hidden\_dim 后面跟tanh激活层

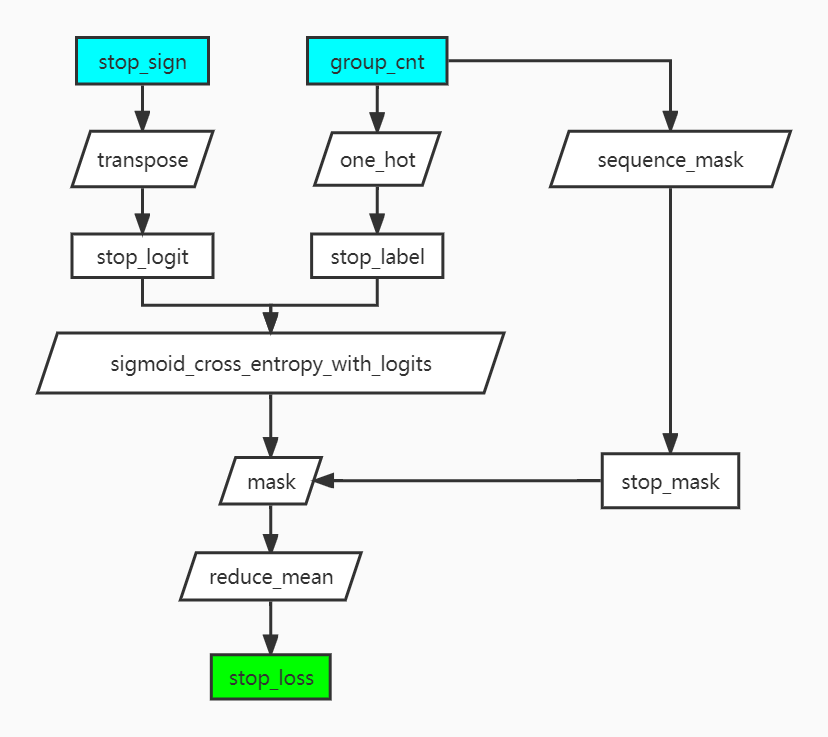
**bow\_fc\_2**: (5.3 parameters) 输出为 tgt\_vocab\_size

**sparse\_soft\_max\_entropy\_with\_logit**: 这里表示的是bow\_logit，预测该seg有哪些词组成，bow\_logit为[batch, seg\_i\_word\_num, tgt\_vocab\_size]。 得到的损失为[batch, seg\_i\_word\_num]，然后通过sent\_mask把该seg多余的空词mask掉，再求和得到[batch,]，利用loss\_mask把空句子mask掉求和得到一个值为bow\_loss。

**输出介绍：**

**bow\_loss**：判断该句子该seg有哪些词的损失，不要求顺序。

#### 5.stop\_loss



**输入介绍：**

**stop\_sign**: (5.4.4 train\_body) 用于记录第i个seg是否停止的标准，最终为[seg\_num, batch]

**group\_cnt**: 原始输入， 记录有多少个seg [batch,]

**层级介绍：**

**transpose**: 更换对应维度，这里为[1,0] 得到stop\_logit [batch, seg\_num]

**one\_hot**: 将group\_cnt进行one\_hot，转成长为seg\_num，stop\_label [batch, seg\_num]

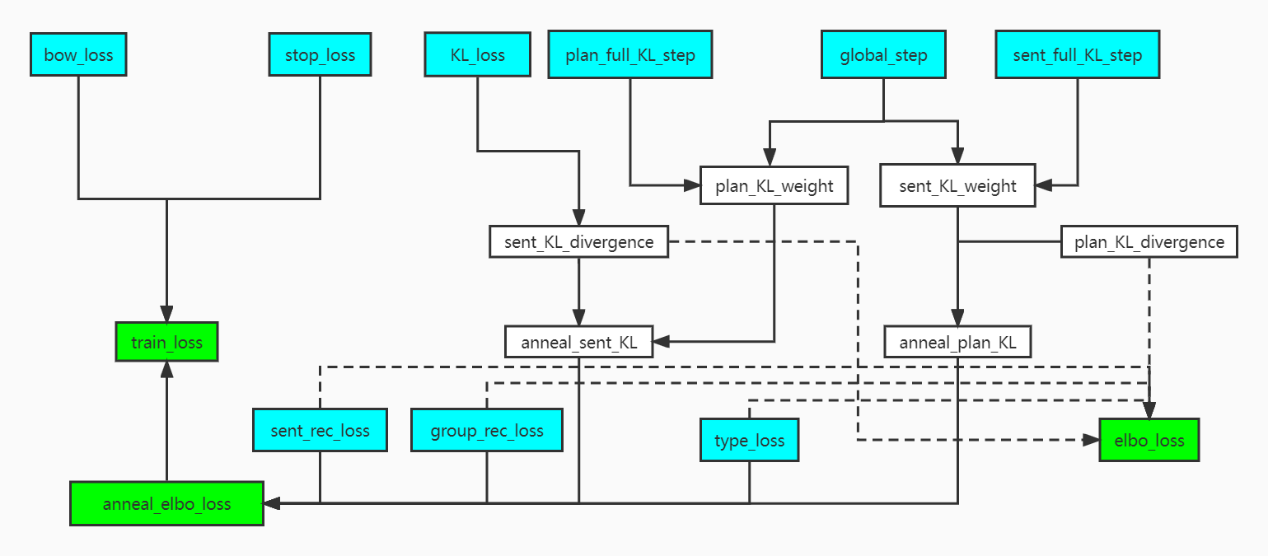
**sigmoid\_cross\_entropy\_with\_logits:** 求stop\_logit和stop\_label的交叉熵，然后通过stop\_mask把空seg的交叉熵mask掉后先求和，得到[batch, ]，再求均值得到一个值stop\_loss。

**sequence\_mask:** 构建mask矩阵，这里将group\_cnt构建成一个长为seg\_num的mask。**stop\_mask**:[batch, seg\_num]

**输出介绍：**

**stop\_loss:**判断指定seg是否停止句子的loss。

#### 6.loss整合



首先介绍经由训练得到的loss：

**KL\_loss:** 即通过kv对和上一句子产生的随机变量和实际句子的信息的随机变量差异，这些随机变量用于产生句子。主要在于是否包含当前seg句子的编码（sent\_embed）和type编码(sent\_type\_embed) KL散度[batch, ]- mask后求和为1个值。

**type\_loss**: 类型损失，判断当前seg有哪些type。 一个值

**sent\_rec\_loss**:是比较生成的句子和实际的句子差异的损失，为单独的一个值

**bow\_loss**：判断该句子该seg有哪些词的损失，不要求顺序。

**group\_rec\_loss**:评估对应句子对应seg中，是否出现对应kv对的损失。group\_label为第i个seg中，有哪些kv对，group\_logit为预测此的结果。float, 累加上所有seg的损失，最终为[batch,]

**plan\_KL\_divergence**:根据大类和句子kv对生成的随机变量和再加上句子生成的随机变量之间的KL损失。

**stop\_loss**:由stop\_sign和实际的seg数group\_cnt求交叉熵得到，最后的值由交叉熵经由group\_cnt的mask后求和得到[batch,]再求均值作为平均每个句子的stop损失，意义为句子需生成多少个seg。

参数：

**global\_step**:为全局迭代次数

**plan\_full\_KL\_step**:用于控制plan\_KL\_divergence损失的权重

**sent\_full\_KL\_step**:用于控制sent\_KL\_divergence损失的权重

计算得到：

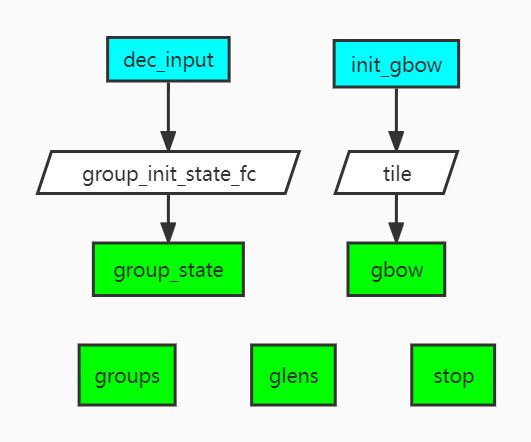
**elbo\_loss**:为sent\_rec\_loss和group\_rec\_loss和type\_loss以及plan\_KL\_divergence和sent\_KL\_divergence。

**anneal\_elbo\_loss**:为添加有权重的elbo\_loss。针对两个KL损失添加权重。

**train\_loss**:anneal\_elbo\_loss和stop\_loss以及bow\_loss的和。

### 5.infer 推断

#### 1.group\_init



**输入介绍：**

**dec\_input:** (4. top level) 不训练时，包含大类的信息，所有kv对的信息，对应的含句子大类和kv对信息的随机变量。[batch, cate\_dim+encoder\_dim(\*2)+plan\_latent\_dim]

**init\_gbow**: (5.3 parameters) 大小为 [1, encoder\_dim(\*2)]的变量

**层级介绍：**

**group\_init\_state\_fc:** (5.3 parameters) 输出group\_decoder\_dim

**tile:** 复制扩展

**输出介绍：**

**group\_state**: 初始化变量 利用dec\_input初始化，包含大类的信息，所有kv对的信息，对应的含句子大类和kv对信息的随机变量 [batch, group\_decoder\_dim]

**gbow**: 初始化变量 [batch, encoder\_dim(\*2)]

**groups:** 初始化变量 [batch, 1, seq\_kv\_num]

**glens**: 初始化变量 [batch,]

**stop**: 初始化变量 [batch,]

#### 2.group\_cond

if i == 0: True

else:

if min(stop) == 0 #该seg所有的句子还存在未停止

if i >= max\_sent\_cnt # seg数不能超过设定的最大seg数

False

else:

True

else:

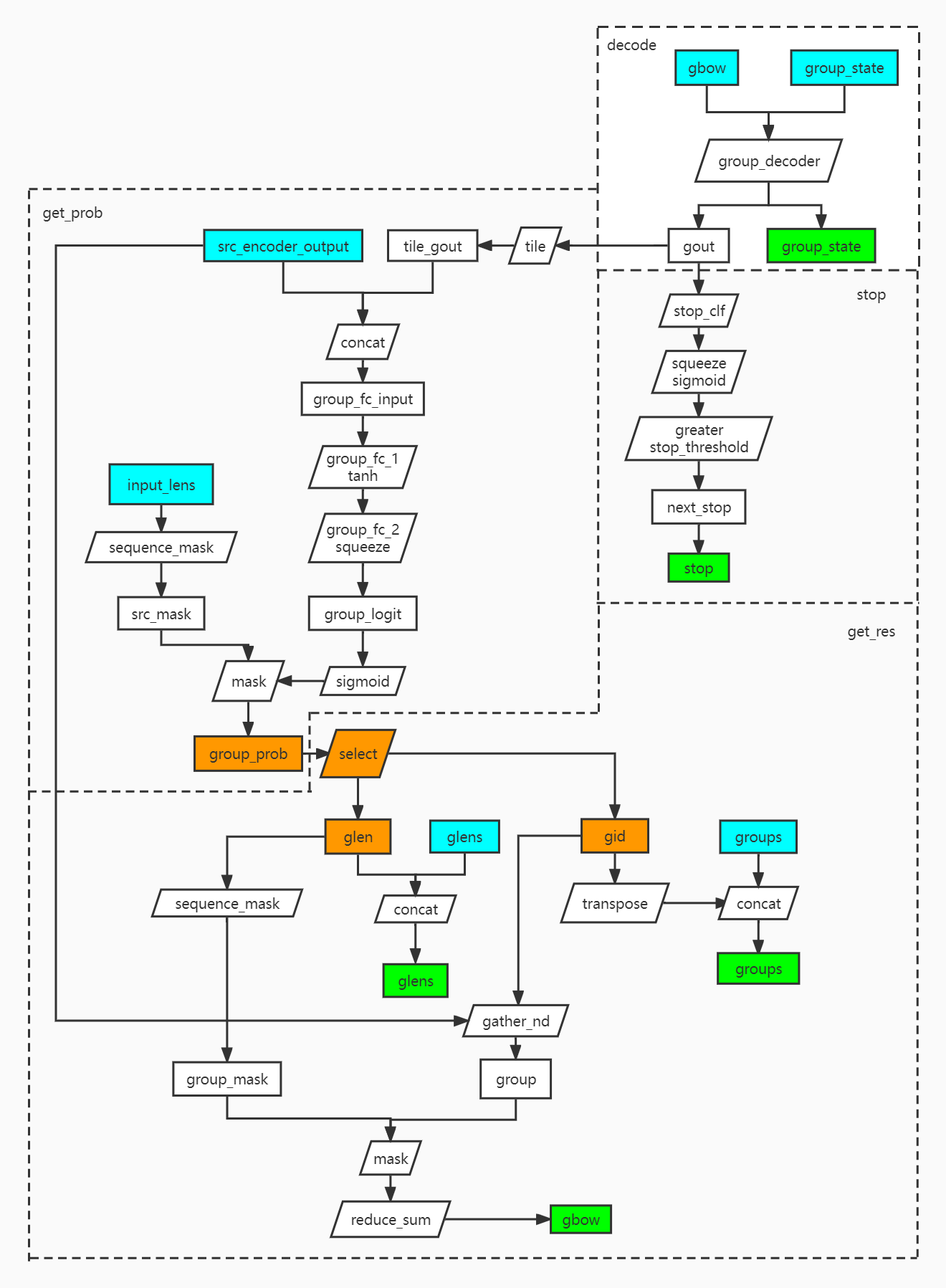
False

最终表示存在句子的seg还能继续往下写才为真

第一步必定为真。

当所有句子都无seg可运行或是seg数超过了设定的最大seg数，就为假

#### 3.group\_body



由于group\_body过于复杂，将其拆分成4部分，decode，stop，get\_porb, get\_res。

分别表示为判断seg有哪些kv对的解码decode，判断第几个seg停止的属性stop，获取对应句子对应seg的kv对出现概率get\_prob，以此得到对应句子seg的kv对的结果和用于下一个seg的kv对编码get\_res。

关键属性：

**group\_state**: 作为输入是前一时刻group\_decoder的状态，输出是当前时刻group\_decoder的状态 [batch, group\_decoder\_dim]

**gbow**: 作为输入是前一时刻生成的seg中含的整体kv对的编码形式。 [batch, encoder\_dim(\*2)]

**groups**: 记录每个句子每个seg的kv对id， [batch, seg\_num, seq\_kv\_num] 值为gid中的seq\_kv\_id。

**glens**: 记录每个句子每个seg的kv对的数量 [batch, seg\_num]

**stop**: 记录每个句子在第几个seg结束 [batch,]

##### 1.decode

**输入介绍：**

**gbow**: 这里表示前一个seg对应的整体kv对的编码形式，值得注意的是，此处为自己求得的gbow [batch, encoder\_dim(\*2)]

**group\_state**: 作为输入是前一时刻group\_decoder的状态，输出是当前时刻group\_decoder的状态。[batch, group\_decoder\_dim]

**层级介绍：**

**group\_decoder:** 为5.1 group\_sent中构建的循环层。输入是指定句子对应seg中整体kv对的编码信息以及decoder上一次的状态，这里由于是针对单独一个seg做运算，gout没有时间步的维度。

**输出介绍：**

**group\_state**: 输出是当前时刻group\_decoder的状态。[batch, group\_decoder\_dim]

**gout**: 包含上一个seg中所有kv对信息解码得到的输出，由group\_decoder得到，[batch, group\_decoder\_dim]

##### 2.stop

**层级介绍：**

**stop\_clf**: (5.3 parameters) 输出1 得到[batch, 1]

**squeeze**: 消除长为1的维度，后面接sigmoid激活 得到 [batch,]

**greater:** 比较大小，超过stop\_threshold的为真 得到next\_stop [batch,]

**输出介绍：**

**stop**: 如果为0的行，表示该句子还没结束，再根据next\_stop，判断下一个seg是否结束，如果下一个seg结束，记当前stop对应行为i+1。

##### 3.get\_porb

**输入介绍：**

**src\_encoder\_output**: (2. input encode) 每一时刻的输出，由于是否双向输出有变化，每一时刻代表每一个kv对，其结果由前面的kv对和后面的kv对共同决定，[batch, seq\_kv\_num, encoder\_dim(\*2)].

**input\_lens**: 原始输入，是整个句子中kv对的数量，取决于原始数据中的feature所表示的所有数量[batch,]

**层级介绍：**

**sequence\_mask**: 将input\_lens化为mask Tensor，长为seq\_kv\_num。 得到src\_mask [batch, seq\_kv\_num]

**group\_fc\_1**: (5.3 parameters) 输出encoder\_dim\*2

**group\_fc\_2:** (5.3 parameters) 输出1

**sigmoid**: 激活层，取消负数的出现

**输出介绍：**

**group\_porb:** 经由group\_logit [batch, seq\_kv\_num]和src\_mask，将对应句子不存在的kv对mask掉后，得到对应句子对应seg出现的kv对的概率，这里之所以是seq\_kv\_num是因为生成的kv对可能是该句子中所有kv对的任意一个，所以跟训练的时候的seg\_i\_kv\_num不一样。[batch, seq\_kv\_num]

##### 4.get\_res

**输入介绍：**

**group\_prob**: 对应句子segi出现的kv对的概率， [batch, seq\_kv\_num]

**glens**: 记录每个句子每个seg的kv对的数量 [batch, seg\_num]

**gid**: 记录对应句子当前seg的kv对编号， [batch, seq\_kv\_num, 2] 最后的2为(batch, seq\_kv\_id)

**groups**: 记录每个句子每个seg的kv对id， [batch, seg\_num, seq\_kv\_num] 值为gid中的seq\_kv\_id。

**glen**: 记录对应句子当前seg的kv对数量 [batch, ]

**层级介绍：**

**select**: 输入为对应句子当前seg出现的kv对的概率，输出为根据概率求得的当前seg出现的kv对的序号，表示为(batch\_id, seq\_kv\_id)，同时输出当前seg出现的kv对的数量，规则为概率超过group\_selection\_threshold的kv对，如果没有，选取最大概率的kv对。

**sequence\_mask:** 将glen进行mask化，得到group\_mask [batch, seq\_kv\_num]的mask矩阵

**gather\_nd:** 根据id将src\_encoder\_output中的kv对编码得到 group[batch, seq\_kv\_num, encoder\_dim(\*2)].

**mask**:用group和group\_mask进行mask后再求和得到gbow:[batch, encoder\_dim(\*2)]

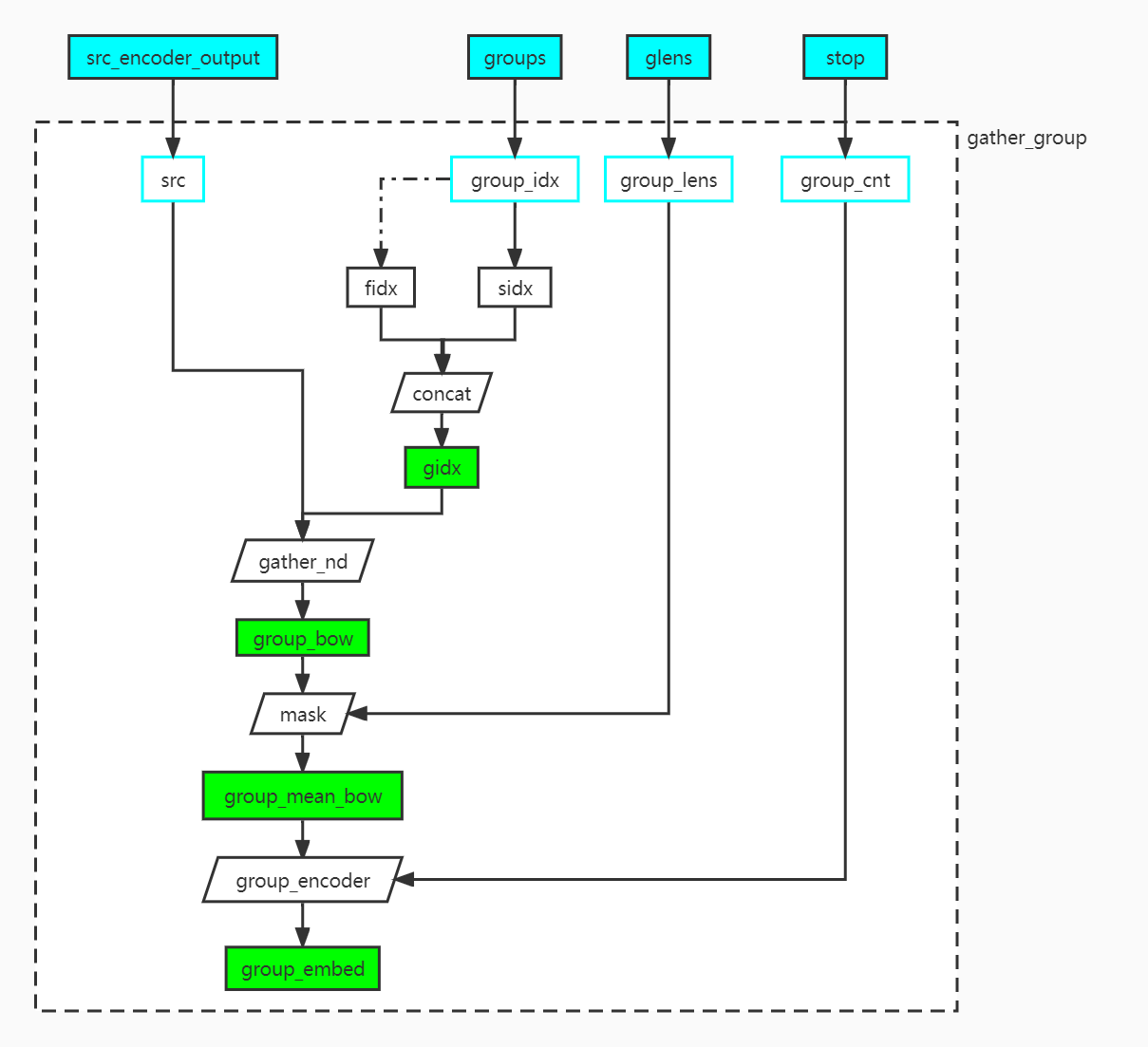
**输出介绍：**

**glens**: 记录每个句子每个seg的kv对的数量 [batch, seg\_num]

**groups**: 记录每个句子每个seg的kv对id， [batch, seg\_num, seq\_kv\_num] 值为gid中的seq\_kv\_id。

**gbow:** 为推断的当前句子当前seg的kv对的整体编码形式 [batch, encoder\_dim(\*2)]

#### 4.group\_encode



**输入介绍：**

**src\_encoder\_output**:出现在 2.input\_encode 处，每一时刻的输出，由于是否双向输出有变化，每一时刻代表每一个kv对，其结果由前面的kv对和后面的kv对共同决定，[batch, seq\_kv\_num, encoder\_dim(\*2)].

**groups**: (5.5.3 group\_body)，记录了每个seg都有哪些kv对，用kv对的编号进行记录，kv对的编号取决于该句子有多少个kv对，即key\_input和val\_input。[batch, seg\_num, seg\_kv\_num]

**glens**: (5.5.3 group\_body)，记录了每个seg的kv对的数量。[batch, seg\_num]

**stop:** (5.5.3 group\_body)，记录了每句话多少个seg。 [batch,]

**层级介绍：**

**gather\_group**: 函数

**蓝框所圈参数**：为函数gather\_group的输入参数，对应输入由图可视。

**concat**:把fidx和sidx的最后一维连接，构成一个序号

**gather\_nd**:依据gidx选取src中对应id的值构成新tensor

**mask**: 根据group\_lens将group\_bow中多余的参数清0

**group\_encoder**:为 5.1 group\_sent 构建的循环层，

**中间变量介绍：**

**fidx**:由group\_idx的形状决定，每个值是对应batch的编号 [batch, seg\_num, seg\_kv\_num, 1]

**sidx**:由group\_idx改变形状得到，每个值对应group的值，[batch, seg\_num, seg\_kv\_num, 1]

**输出介绍：**

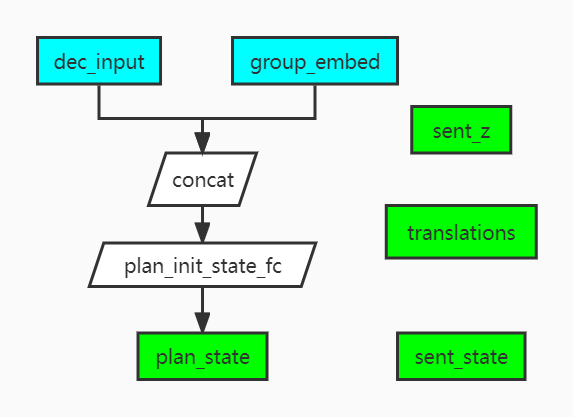
**gidx**:由fidx,sidx连接得到，每个值对应为(batch, seq\_kv\_id)的含义，即序号，序号取决于对应句子kv对的编号，记录每个句子每个seg对应的kv对的序号。[batch, seg\_num, seg\_kv\_num, 2]。

**group\_bow**:对应句子对应seg的对应kv对的编码形式。一个kv对对应一个编码，[batch, seg\_num, seg\_kv\_num, encoder\_dim(\*2)]。

**group\_mean\_bow**:对应句子对应seg的整体kv对的编码形式。这里将所有kv对的编码向量求均值得到，一个seg对应一个编码[batch, seg\_num, encoder\_dim(\*2)]

**group\_embed**:输入的time\_step为每句话的seg，值是每句话对应seg的整体kv对的编码形式，这里将所有kv对的编码向量求均值得到，经过循环神经网络得到最后一个时间步的输出。[batch, group\_decoder\_dim]。

#### 5.infer\_init



**输入介绍：**

**dec\_input**: 4.top\_level中的输出，训练时，为cate\_embed,src\_embed,post\_z\_plan的连接，

否则为cate\_embed,src\_embed,prior\_z\_plan。训练时，包含大类的信息，所有kv对的信息，对应的含句子大类和kv对以及句子本身的信息的随机变量（变分思想）；不训练时，包含大类的信息，所有kv对的信息，对应的含句子大类和kv对信息的随机变量。[batch, cate\_dim+encoder\_dim(\*2)+plan\_latent\_dim]

**group\_embed**: 5.5.4 group\_encode输入的time\_step为每句话的seg，值是每句话对应seg的整体kv对的编码形式，这里将所有kv对的编码向量求均值得到，经过循环神经网络得到最后一个时间步的输出。[batch, group\_decoder\_dim]。

**层级介绍：**

**concat**: 连接

**plan\_init\_state\_fc**: 5.3 parameters 输出latent\_decoder\_dim

**输出介绍：**

**sent\_z:** 初始化用于送入循环体的sent\_z，初始化为0，[batch, sent\_latent\_dim]

**translations:** 初始化用于送入循环体的translations，初始化为- [batch, 1, maxmum\_iterations]

**sent\_state**: 初始化decoder的状态，

#### 6. infer\_cond

i<seg\_num， 生成句子的seg数不超过seg\_num。

#### 7.infer\_body

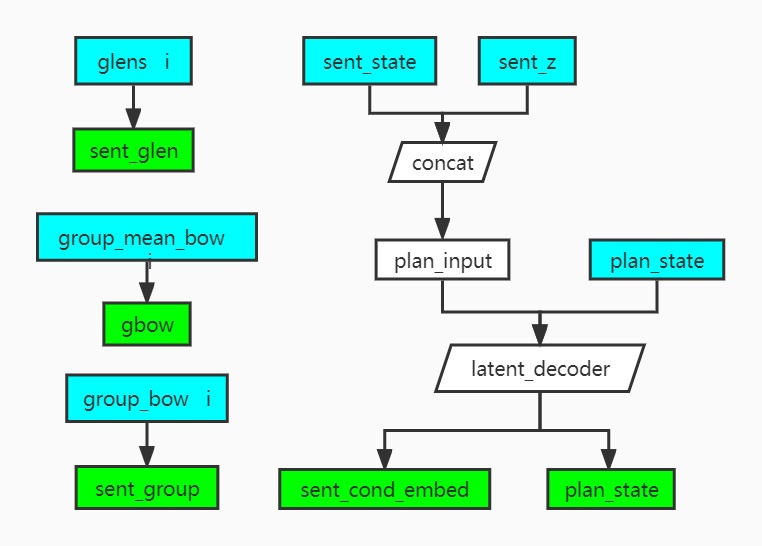
plan\_state: latent\_decoder上一时刻的状态 [batch, decoder\_dim]

sent\_state:

sent\_z: 经历重采样得到的随机变量信息，包含整体kv对编码，整体type编码的信息和之前状态的信息。[batch, sent\_latent\_dim]

translations:

##### 1.plan and init



**输入介绍：**

**glens:** (5.5.3 group\_body)记录每个句子每个seg的kv对的数量 [batch, seg\_num]

**sent\_state**: 为basic\_decoder解码后的状态，即解码出来当前seg句子后的decoder状态。[batch, decoder\_dim]

**sent\_z:** 经历重采样得到的随机变量信息，包含整体kv对编码，整体type编码的信息和之前状态的信息。[batch, sent\_latent\_dim]

**group\_mean\_bow**: (5.5.4 group\_encode) 对应句子对应seg的整体kv对的编码形式。这里将所有kv对的编码向量求均值得到，一个seg对应一个编码[batch, seg\_num, encoder\_dim(\*2)]

**group\_bow**: (5.5.4 group\_encode) 对应句子对应seg的对应kv对的编码形式。一个kv对对应一个编码，[batch, seg\_num, seg\_kv\_num, encoder\_dim(\*2)]。

**plan\_state:** latent\_decoder上一时刻的状态 [batch, decoder\_dim]

**层级介绍：**

**latent\_decoder**: （5.1 group\_sent） 输出 latent\_decoder\_dim

**输出介绍：**

**sent\_glen**: 指定句子该seg有多少kv对 [batch,]

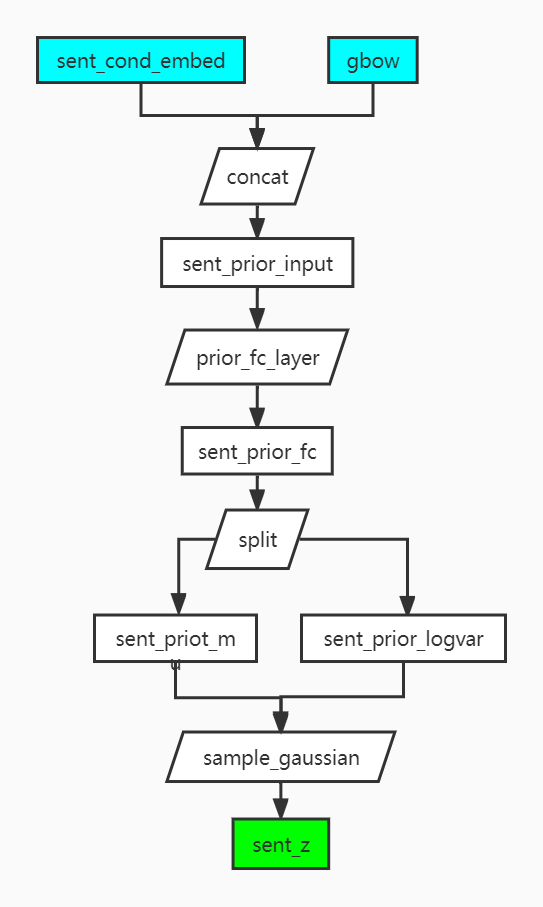
**gbow**: 指定句子第i个seg的整体kv对编码形式 [batch, encoder\_dim(\*2)]

**sent\_group**: 指定句子指定seg的每个kv对的编码形式 [batch, seg\_i\_kv\_num, encoder\_dim(\*2)]

**sent\_cond\_embed**: 来自sent\_state和sent\_z的信息，加上latent\_decoder之前的状态解码得到的输出[batch, latent\_decoder\_dim]

**plan\_state**: latent\_decoder当前时刻的状态，主要用于seg传递decoder状态，[batch, latent\_decoder\_dim]

##### 2.resample



**输入介绍**

**sent\_cond\_embed**： (5.5.7.1plan and init)来自sent\_state和sent\_z的信息，加上latent\_decoder之前的状态解码得到的输出[batch, latent\_decoder\_dim]

**gbow:** (5.5.7.1plan and init)指定句子第i个seg的整体kv对编码形式 [batch, encoder\_dim(\*2)]

**层级介绍：**

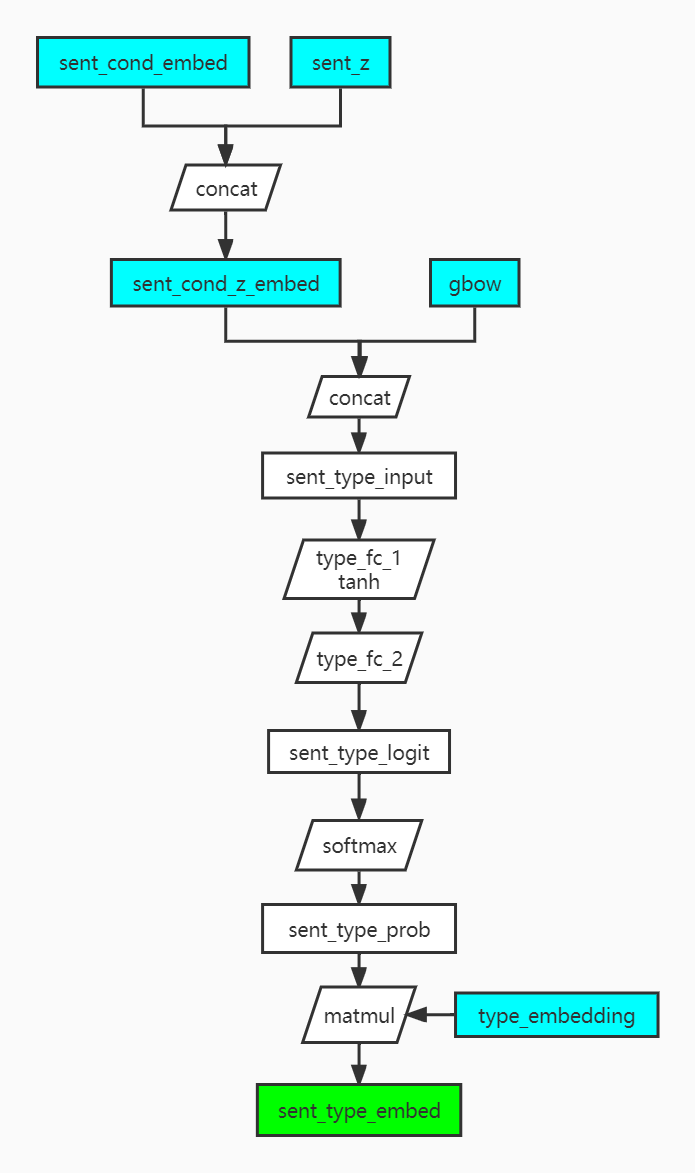
**prior\_fc\_layer:** （5.3 parameters）输出 sent\_latent\_dim\*2

sample\_gaussian: 重采样

**输出介绍：**

**sent\_z**: 经历重采样得到的随机变量信息，包含整体kv对编码，整体type编码的信息和之前状态的信息。[batch, sent\_latent\_dim]

##### 3.type



**输入介绍：**

**sent\_cond\_embed**: （5.5.7.1 plan and init）来自sent\_state和sent\_z的信息，加上latent\_decoder之前的状态解码得到的输出[batch, latent\_decoder\_dim]

**sent\_z:** （5.5.7.2 resample） 经历重采样得到的随机变量信息，包含整体kv对编码，整体type编码的信息和之前状态的信息。[batch, sent\_latent\_dim]

**sent\_cond\_z\_embed**: 由sent\_cond\_embed和sent\_z连接而得，包含上一状态的信息，和当前整体kv对和整体type编码的信息， [batch, latent\_decoder\_dim + sent\_latent\_dim]

**gbow:** (5.5.7.1plan and init)指定句子第i个seg的整体kv对编码形式 [batch, encoder\_dim(\*2)]

**层级介绍：**

**type\_embedding**: 1embedding 类相关的id的embedding,embedding到type\_dim。

**type\_fc\_1**: 5.3 parameters 输出type\_dim

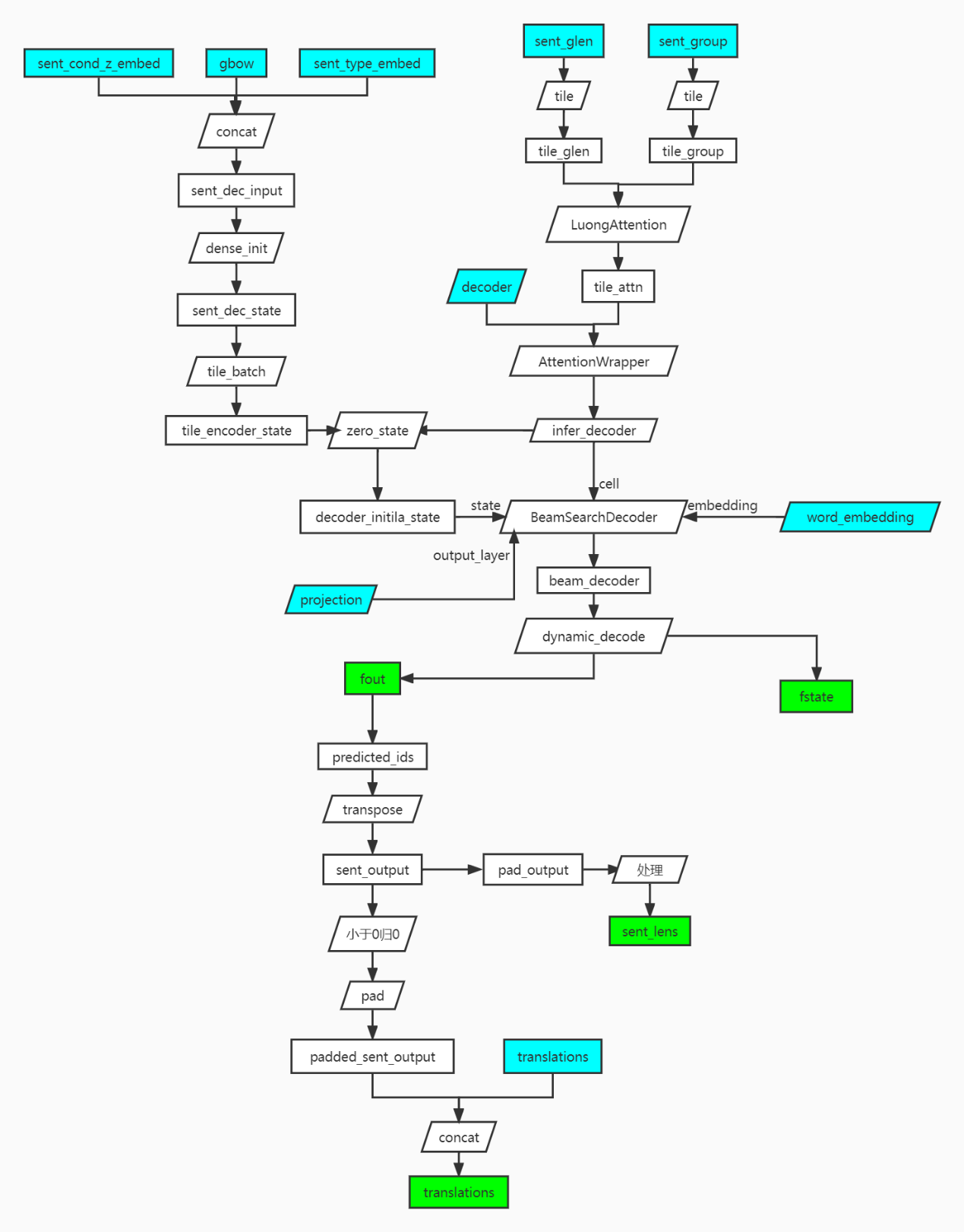
**type\_fc\_2**: 5.3 parameters 输出type\_vocab\_size

**matmul**: 矩阵相乘得到sent\_type\_embed

**输出介绍：**

**sent\_type\_embed**: 推测的seg的type的embed形式 [batch, type\_dim]

##### 4.sent decode



**输入介绍：**

**sent\_cond\_z\_embed**: 由sent\_cond\_embed和sent\_z连接而得，包含上一状态的信息，和当前整体kv对和整体type编码的信息， [batch, latent\_decoder\_dim + sent\_latent\_dim]

**gbow:** (5.5.7.1plan and init)指定句子第i个seg的整体kv对编码形式 [batch, encoder\_dim(\*2)]

**sent\_type\_embed:** (5.5.7.3 type)推测的seg的type的embed形式 [batch, type\_dim]

**sent\_glen:** 指定句子该seg有多少kv对 [batch,]

**sent\_group:** 指定句子指定seg的每个kv对的编码形式 [batch, seg\_i\_kv\_num, encoder\_dim(\*2)]

**decoder**: 5.1 group\_sent 输出decoder\_dim

**word\_embedding**: 1.embedding target相关的id的embedding，embedding到word\_dim

**projection**: 5.3 parameters 输出tgt\_vocab\_size

**层级介绍：**

**sent\_dec\_input**: 由上一状态的sent信息(sent\_state&sent\_z)和当前状态产生的sent信息得到,包含有seg句子信息，整体kv对编码，type编码，上一时刻的sent信息。以及直接包含的整体kv对编码信息和整体type编码信息。[batch, latent\_decoder\_dim + encoder\_dim +type\_dim]

**dense\_init**: 将sent\_cond\_z\_embed，gbow和sent\_type\_embed连接后得到sent\_dec\_input，通过自动随机生成dense层的参数，使sent\_dec\_input通过dense初始化为sent\_dec\_state。使得sent\_dec\_state初始化含上一状态信息，当前sent重采样信息(sent\_z)，整体kv对信息，type编码信息，使得decoder的状态更接近实际。

**tile\_batch:** 此处用于添加beam\_size的纬度用于BeamSearchDecoder的状态初始化，得到tile\_encoder\_state。 [batch\*beam\_witdh, decoder\_dim]

**zero\_state:**初始化decoder状态

**tile:**添加beam\_width用于attention的构建

**infer\_decoder:**添加了attention的decoder。

**BeamSearchDecoder**:用beam\_search的方法解码，用infer\_decoder作为decoder，decoder\_initle\_state为初始化状态，word\_embedding为编码, 选取最有可能的序列，不再使用贪心求解。

**predicted\_ids:** 预测的为beam\_size下的句子 [batch, seg\_i\_kv\_num, beam\_size] 这里seg\_i\_kv\_num其实只是单纯的由于输入设置的，其实质这里的含义应为 seg\_i\_word\_num。

**sent\_output:**选取的对应seg最大的概率的句子 [batch, seg\_i\_word\_num]

**小于0归0：**使得sent\_output不存在未知符

**处理：**将sent\_lens变为句子长度 [batch, ]

**输出介绍：**

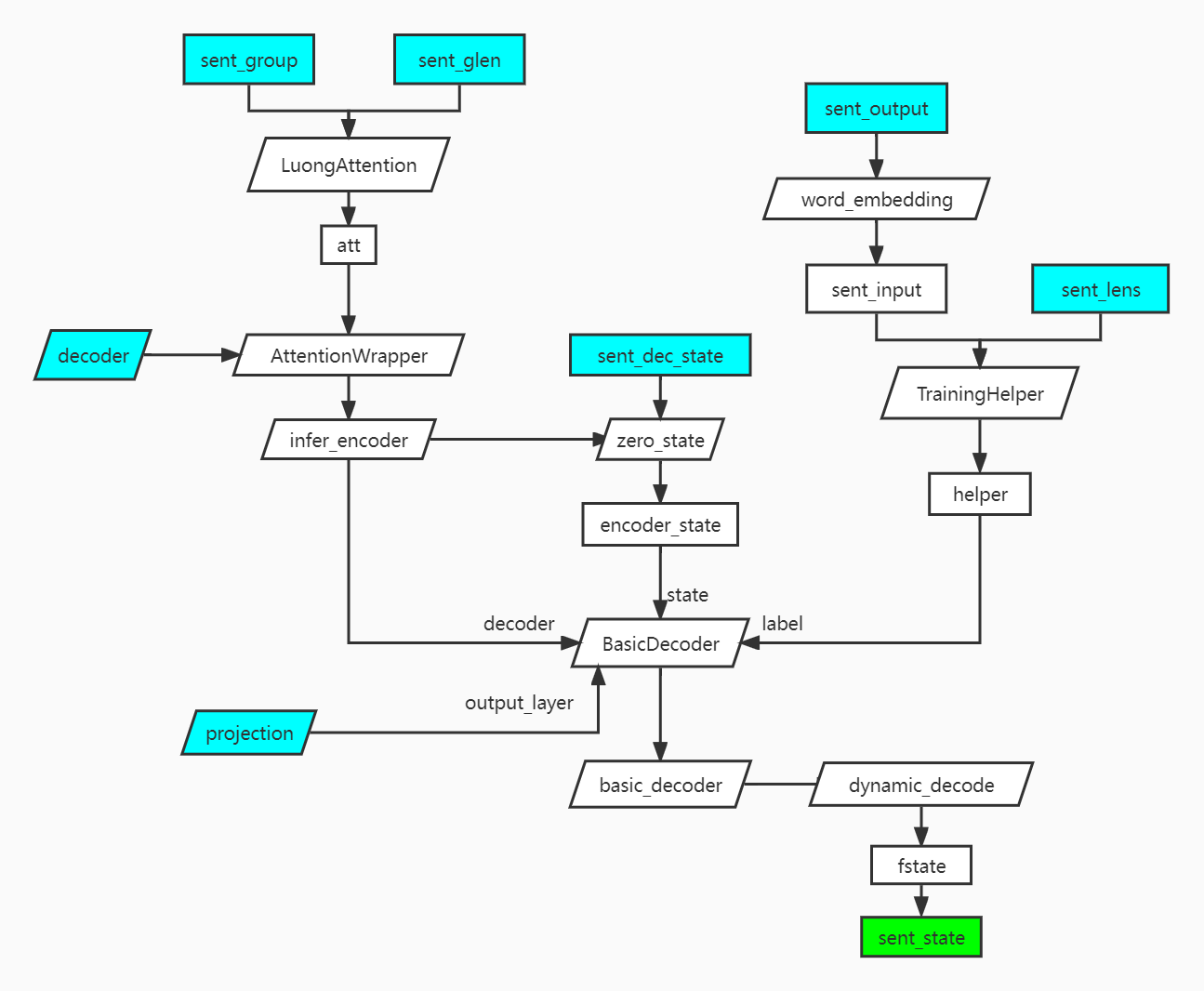
**fstate**: decoder的下一个状态

**sent\_lens:** 对应seg句子的长度 [batch, ]

**translations:** 记录下每个seg生成的句子 [batch, seg\_num, maxiterations]

##### 5.encode

用于下一状态的编码



输入介绍：

sent\_group: 指定句子指定seg的每个kv对的编码形式 [batch, seg\_i\_kv\_num, encoder\_dim(\*2)]

sent\_glen: 指定句子该seg有多少kv对 [batch,]

sent\_output: 选取的对应seg最大的概率的句子 [batch, seg\_i\_word\_num]

decoder: 5.1 group\_sent 输出decoder\_dim

sent\_dec\_state: 将sent\_cond\_z\_embed，gbow和sent\_type\_embed连接后得到sent\_dec\_input，通过自动随机生成dense层的参数，使sent\_dec\_input通过dense初始化为sent\_dec\_state。使得sent\_dec\_state初始化含上一状态信息，当前sent重采样信息(sent\_z)，整体kv对信息，type编码信息，使得decoder的状态更接近实际

sent\_lens: 对应seg句子的长度 [batch, ]

projection: 5.3 parameters 输出tgt\_vocab\_size

输出介绍：

sent\_state: 下一刻句子的状态 [batch, decoder\_dim]