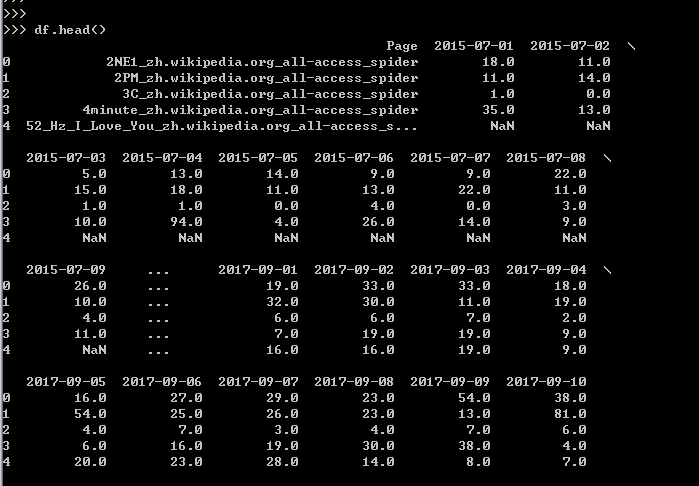
# 一、数据结构

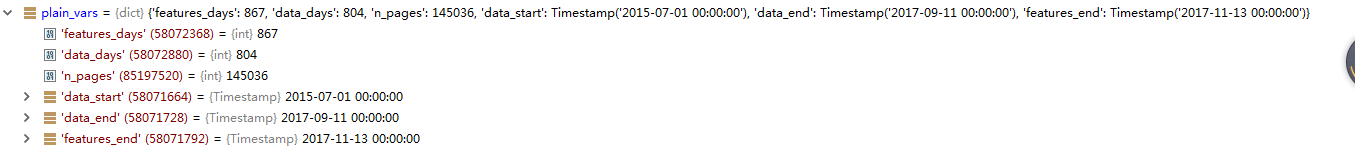
## 1. 原始数据结构



## 2.基本特征结构

包含两个字典变量 **tensor\_vars（训练数据）、tensor\_vars （一些标量）**





1. **tensor\_vars:（11个）**

hits:原始时间序列(145036, 804) ，804天，不包含预测未来的63天

lagged\_ix :(867,4) 3, 6, 9, 12 月前销量的序号 0---804，前面没有的用-1代替

page\_map:(52752,4) 唯一url，四个代理

例如：

1 2 -1 0

4 5 -1 3 ...

page\_ix: (145036) 原始url

pf\_agent:(145036,4) 每行四个值，分别代表四个代理，onhot后标准化的值

例如：1.63734889912 -0.562919854285 -0.561804678048 -0.573951364895

-0.61073916848 1.77643957225 -0.561804678048 -0.573951364895

pf\_country:(145036,4) 每行7个值，分别代表7个国家，onhot后标准化的值

pf\_site : (145036,3) 每行3个值，分别代表3个站点，onhot后标准化的值

page\_popularity: (145036) 每个url销量求中位数，计算一个流行度，然后标准化

* page\_popularity = df.median(axis=1)
* page\_popularity = (page\_popularity - page\_popularity.mean()) /page\_popularity.std()

year\_autocorr: (145036)年度自相关系数，也是标准化成 0均值，1方差

* corr[i] = 0.5 \* c\_365 + 0.25 \* c\_364 + 0.25 \* c\_366

quarter\_autocorr: (145036) 季度自相关系数 365.25 / 4 =91 ，也是标准化成 0均值，1方差

* corr[i] = 0.5 \* c\_90 + 0.25 \* c\_91 + 0.25 \* c\_92

dow: (867,2) 时间序列周（0-6）的余弦、正弦值 （-1，-1），包含预测的未来67天

1. **plain\_vars:（6个）**

features\_days=867 (2015-07-01, 2017-11-13) 特征起始结束时间（包含预测时间段）

data\_days=804 (2015-07-01, 2017-09-11) 训练数据起始结束时间（不包含预测）

n\_pages=145036 一共多少行数据

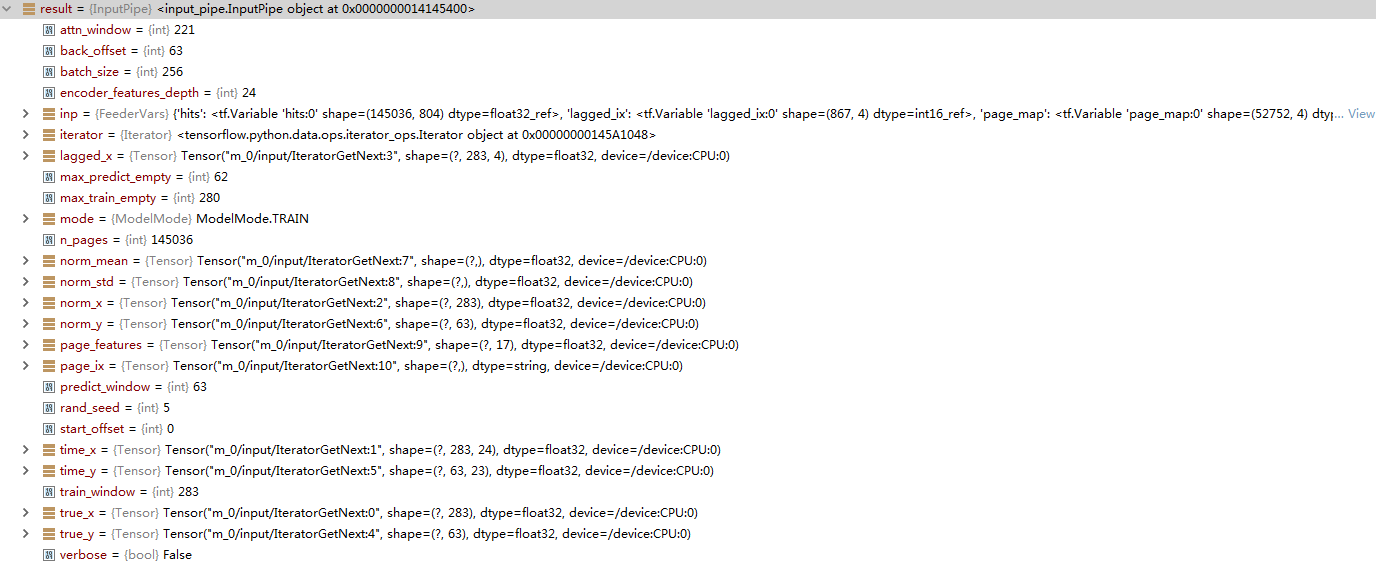
data\_start= 2015-07-01 数据起始时间

data\_end= 2017-09-11 数据结束时间

features\_end=2017-11-13 特征结束时间（预测时段最后一天）

# 二、tensor结构 inp

Tensor结构是用来模型训练的结构，是在基本特征的基础上略就加工后的结果。



true\_x：(?,283) 283天的销量

true\_x：(?,63) 63天的销量 (真实label)

time\_x：(?, 283,24) 包含真实销量

norm\_x\_hits(283,1)+x\_dow(283,2)+x\_lagged(283,4)+tf.tile(page\_features, [self.train\_window, 1])(283,16)

其中:page\_features:pf\_agent, pf\_country, pf\_site, page\_popularity, quarter\_autocorr, year\_autocorr (6个没有page\_ix）

time\_y:(63,23)

y\_dow(63,2)+y\_lagged(63,4)+tf.tile(page\_features, [self.predict\_window, 1])(63,17)

# 三、模型

**用283天预测63: seq2seq seq(283)-to-sqe(63)**

## 1.模型结构图

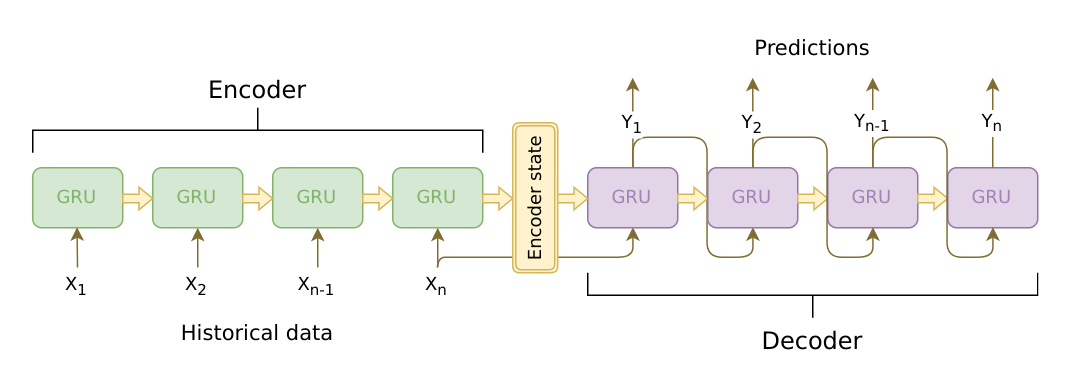


图1 seq2seq原理



图2 神经网络总体结构



图 3 encoder流程



图 4 生成指纹（fingerprint）流程



图 5 生成注意力特征流程



图 6 decoder流程

## 2. 模型代码摘要

1). make\_encoder：

输入：

time\_x:*[batch, time, features] ==（？，283，24）*

*处理：*

transpose：*[time, batch, features] ==（283，？，24）*

*输出：*

encoder\_output:*[time, batch,* rnn\_depth*] ==（283，？，267）*

h\_state=*[1, batch,* rnn\_depth*] ==（1，？，267）*

2). convert\_cudnn\_state\_v2：

输入：

h\_state：*(1,batch,*rnn\_depth*)=（1，？，267）*

*处理：*

*dropout: (1，？，267）*

*输出：*

encoder\_state：*(1，？，267）*

3). compressed\_readout:

输入：

encoder\_output:*[time, batch,* rnn\_depth*] ==（283，？，267）*

*处理：*

*dropout: (283，？，267）*

*输出：*

enc\_readout: *[time, batch,* attention\_depth*] ==（283，？，64）*

4). fingerprint\_inp:

输入：

fingerprint\_inp=lagged\_x([batch,time,features]==(?,283,4))+norm\_x(?,283,1)

=(?,283,5)

处理：

c11 = conv1d(x, filters=16, kernel\_size=7) # ==(?,283,16)

c12 = conv1d(c11, filters=16, kernel\_size=3) # ==(?,283,16)

pool1 =max\_pooling1d(c12, 2, 2) # ==(?,142,16)

c21 = conv1d(pool1, filters=32, kernel\_size=3)# ==(?,142,32)

c22 = conv1d(c21, filters=32, kernel\_size=3)# ==(?,142,32)

pool2 = max\_pooling1d(c22, 2, 2)# ==(?,71,32)

c31 = conv1d(pool2, filters=64, kernel\_size=3)# ==(?,71,64)

c32 = conv1d(c31, filters=64, kernel\_size=3)# ==(?,71,64)

pool3 = max\_pooling1d(c32, 2, 2)# ==(?,36,64)

pool3 = tf.reshape(pool3, [-1, 36 \* 64]) #== (?,2304)

cnn\_out = dropout(pool3) #== (?,2304)

fc\_encoder = dense(cnn\_out, 512) # ==(?,512)

输出：

fingerprint= dense(fc\_encoder, 16)# ==(?,16)

5). attn\_readout\_v3:

输入：

enc\_readout:(*283，？，64）*

fingerprint ：(?,16)

处理：

*#[n\_days, batch, readout\_depth] -> [readout\_depth, width=n\_days, channels=batch]*

readout = tf.transpose(enc\_readout, [2, 0, 1]) # (64,283,?)

*# [batch(readout\_depth), width, channels] -> [batch, height=1, width, channels]*

inp = readout[:, tf.newaxis, :, :] # (64,1,283,?)

# *[batch, attn\_window \* n\_heads]* (?,221)

filter\_logits = dense(fingerprint, attn\_window\*attn\_heads)

#*[batch, attn\_window \* n\_heads] -> [batch, attn\_window, n\_heads]==*(?,221,1)

filter\_logits = reshape(filter\_logits, [-1, attn\_window, attn\_heads])

# 与最大值比值 (?,221,1)

attns\_max = filter\_logits / tf.reduce\_sum(filter\_logits, axis=1)

# *[batch, attn\_window, n\_heads] -> [width(attn\_window), channels(batch), n\_heads] ==(221,?,1)*

attns\_max = tf.transpose(attns\_max, [1, 0, 2])

*# [width(attn\_window), channels(batch), n\_heads] -> [height(1), width(attn\_window), channels(batch), multiplier(n\_heads)] =(1,221,?,1)*

attn\_filter = attns\_max[tf.newaxis, :, :, :]

# [batch(readout\_depth), height=1, width=n\_days, channels=batch] -> [batch(readout\_depth), height=1, width=predict\_window, channels=batch\*n\_heads] --(64,1,63,?) （64，1，283，？） 用卷积核（1，221，？，1）做深度卷积 生成（64，1，63，？）

averaged = tf.nn.depthwise\_conv2d\_native(inp, attn\_filter, [1, 1, 1, 1], **'VALID'**)

attn\_features = tf.squeeze(averaged, 1) #(64,63,?)

*# [batch(depth), predict\_window, batch\*n\_heads] -> [batch\*n\_heads, predict\_window, depth] ==(?,63,64)*

attn\_features = tf.transpose(attn\_features, [2, 1, 0])

*# [batch \* n\_heads, predict\_window, depth] -> n\_heads \* [batch, predict\_window, depth] [(?,63,64)]*

heads = [attn\_features[head\_no::attn\_heads] **for** head\_no **in** range(attn\_heads)]

输出：

*# n\_heads \* [batch, predict\_window, depth] -> [batch, predict\_window, depth\*n\_heads] ==(?,63,64\*1)*

attn\_features= tf.concat(heads, axis=-1)

6). decoder

输入：

encoder\_state：*(1，？，267）--*squeeze（？，267）

attn\_features：*(?,63,64\*1)*

time\_y（prediction\_inputs）：（？,63,23) 未来的已知特征

norm\_x[:, -1]（previous\_y）:(?,283)--(?) 预测前最后一天真实销量

处理：

***# [batch\_size, time, input\_depth] -> [time, batch\_size, input\_depth] (63,?,23) 改成时间维度在前，跟训练保持一致***

inputs\_by\_time = tf.transpose(time\_y, [1, 0, 2])

features =inputs\_by\_time[time] #（?,23)

初始输入：

prev\_state =encoder\_state

prev\_output =previous\_y

开始循环：

**# prev\_output：*(?,1)* features = inputs\_by\_time[time]：（？，23） attn = attn\_features[:, time, :]：（？，64）**

next\_input = tf.concat([prev\_output, features, attn], axis=1) # *==(?,88)*

output, state = cell(next\_input, prev\_state) # 输出下一步 *(?,267)-(?,267)*

projected\_output = project\_output(output) # 生成1维度结果(?,1)

prev\_output=projected\_output

prev\_state =state

# 结束循环

array\_targets = array\_targets.write(time, projected\_output) #放入结果Array

输出：

*# [time, batch\_size, 1] -> [time, batch\_size] ==（63，？）*

targets = tf.squeeze(targets, axis=-1)

7). decode\_predictions

输入：decoder\_targets（*63，？*）

处理：

*# [n\_days, batch] -> [batch, n\_days]* （*63，？*）--（*？，63*）

batch\_readout = tf.transpose(decoder\_readout)

# 反标准化 （*63，？*）

predictions =batch\_readout \* norm\_std+ norm\_mean

输出：

predictions：（*63，？*）