

MRSA论文复现结果

NPRA

基本模型的理解

Neural Recommendation with Personalized Attention

创新点为个性化的attention

- 使用word embedding对每一句review进行处理，将每句review产生的矩阵经过cnn。
- cnn每一列代表了review中一个词的特征
- 考虑每个用户/产品对一句话中每一个词有不同的关注度，对每个用户/产品生成一个attention vector
- 根据attention vector产生一个个性化的关注度权重，对词特征矩阵进行加权平均得到某个针对user/item的review表示向量
- 一个用户/产品有很多review，每条review都有自己的表示向量。整合所有review的表示向量
- 考虑每个用户/产品对每条review也有关注的不同，再次使用个性化的attention产生用户/产品对每条review的attention weight，加权平均得到一个用户/产品的表示向量
- 对每一对用户--产品表示向量，经过FM能得到用户与产品匹配的rate

与上一次实现的不同/优化

使用yelp13数据集

实现了word embedding

word embedding 实际上是一个网络，输入为所选词的下标，输出为词向量，参数为一个weight矩阵。

用户id/产品id没有语义，使用了随机weight矩阵。

review中word是有语义特征的，使用google word2vec产生weight矩阵

主要代码为

```
word-embedding.py
```

```

for i, word in enumerate(target_vocab):
    try:
        weights_matrix_user[i] = gensim_model[word]
        weights_matrix_item[i] = gensim_model[word]
        words_found += 1
    except KeyError:
        weights_matrix_user[i] = np.random.normal(scale=0.6, size=
(WORD_EMBEDDING_DIM, ))
        weights_matrix_item[i] = np.random.normal(scale=0.6, size=
(WORD_EMBEDDING_DIM, ))

```

model部分

使用了padding将review个数，review中的词数填写到定长 WORDS_SIZE / REVIEWS_SIZE

在确定padding长度时，发现 words_size/review_size 分布为中间多两头小（可能是正态），取最大值有些不妥

cal_padding.py 几个计算函数

```

def count_word_size():
    ...

# sentence_level predict
def count_sent_size():
    ...

# review_level dim predict

def count_review_num(dict):
    ...

```

输出结果如下：

```

-----word size-----
100% 1320
95%: 480
90%: 379
85%: 137

```

```
----- user -----
allnum: 1631
95%: 91
90%: 69
85%: 56
----- item -----
allnum: 1633
95%: 90
90%: 69
85%: 57
```

最终确定wordsize为379 user/item_review_size为69

修改model forward中循环计算为reshape 矩阵到子网络中 输出后再reshape到期望大小

model代码文件为 `nrpa.py`

数据集

主要思路是产生username、itemname、vocab、user_reviws、item_reviews字典查表产生数据集

主要代码为 `data.py`

主要函数

- `init_id_dict`
- `init_voc_dict`
- `review_devide_by_user`
- `review_devide_by_item`
- `NRPAdataset` 类:
 - `get_vec_of_review`
 - `get_review_tensor_from_list`

惰性加载

在第一次实现时，选择吧user_all_review向量加载到内存，`__getitem__` 简单取下标就行。但是初始化太慢

改为，将user的review下标(train.txt中的行数)存储为一个列表，在 `__getitem__` 时再展开为tensor

自定义dataloader

```
def collate_fn(data):
    # user_review_vectors, user_id_vector, item_review_vectors, item_id_vector
    data = list(zip(*data))

    rslt_user_review = torch.stack(data[0], dim=0)
    rslt_user_id = torch.stack(data[1], dim=0)
    rslt_item_review = torch.stack(data[2], dim=0)
    rslt_item_id = torch.stack(data[3], dim=0)
    rslt_rate = torch.stack(data[4], dim=0)

    return rslt_user_review, rslt_user_id, rslt_item_review, rslt_item_id,
    rslt_rate
```

最终进度

模型跑起来，但是学小container昨天12点被销毁了，2点重申请到一个比较慢的gpu...

跑了两个epoch。

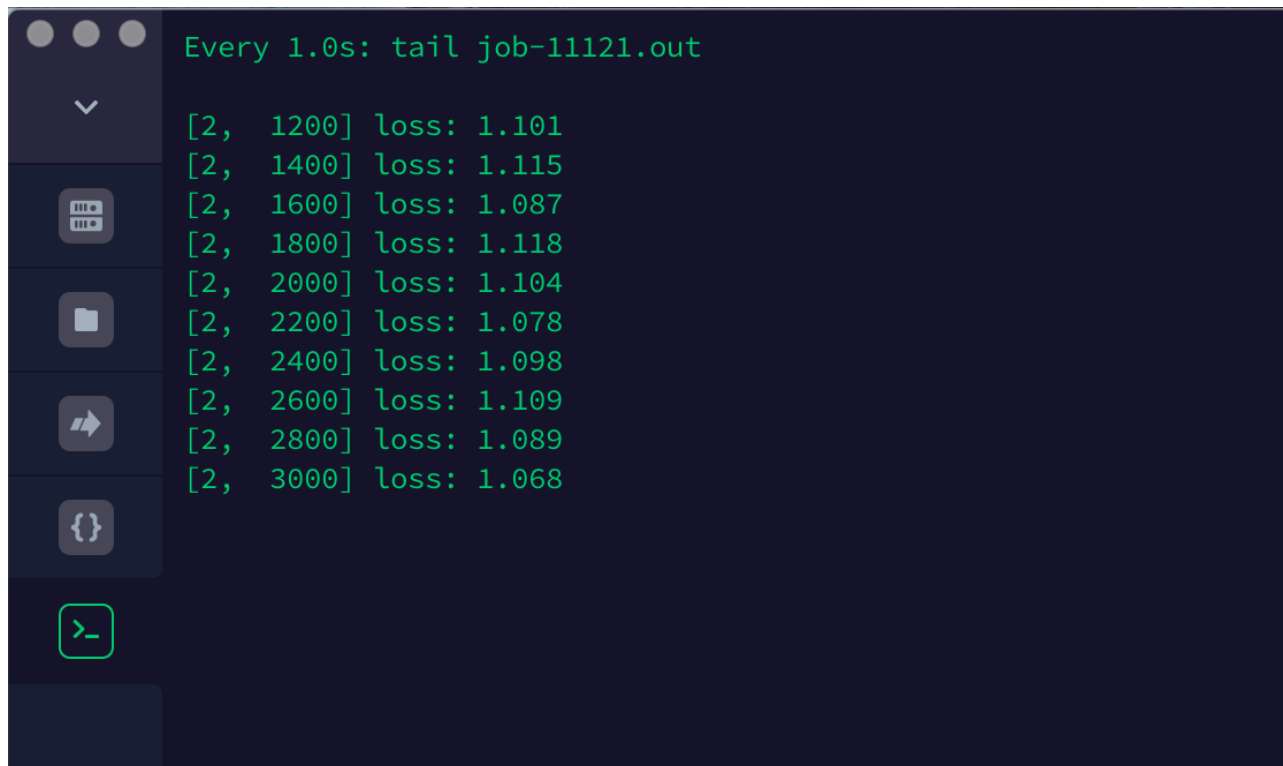
第一次跑起来的bias和V和W应该要非常小，否则输出会非常大。导致loss巨大

初始化FM的参数到0/1

```
self.linear.load_state_dict({"weight": torch.zeros(1, input_dim), "bias":
torch.ones(1)})
self.V = nn.Parameter(torch.zeros(input_dim, input_dim))
```

效果如下:

```
root@35b3ce2044e3:~/weiruan/nrpa# python train.py
/opt/conda/lib/python3.6/site-packages/sklearn/externals/joblib/externals/cloudpickle/cloudpickle.py:47: DeprecationWarning: the imp module is deprecated in favour of importlib; see the module's documentation for alternative uses
  import imp
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
----- finish data load -----
----- finish net load -----
----- start training -----
[1, 200] loss: 1.508
[1, 400] loss: 1.105
[1, 600] loss: 1.086
[1, 800] loss: 1.100
[1, 1000] loss: 1.175
[1, 1200] loss: 1.176
[1, 1400] loss: 1.134
[1, 1600] loss: 1.127
[1, 1800] loss: 1.097
[1, 2000] loss: 1.106
[1, 2200] loss: 1.110
[1, 2400] loss: 1.120
[1, 2600] loss: 1.126
[1, 2800] loss: 1.116
[1, 3000] loss: 1.086
[2, 200] loss: 1.826
[2, 400] loss: 1.111
[2, 600] loss: 1.146
[2, 800] loss: 1.147
[2, 1000] loss: 1.096
[2, 1200] loss: 1.122
[2, 1400] loss: 1.106
```

A terminal window with a dark blue background and light green text. The title bar shows three window control buttons (red, yellow, green) on the left. Below the title bar is a sidebar with icons for a dropdown menu, a server rack, a folder, a right-pointing arrow, a code block, and a terminal. The main area of the terminal displays the command 'Every 1.0s: tail job-11121.out' followed by a series of lines showing training progress: '[2, 1200] loss: 1.101', '[2, 1400] loss: 1.115', '[2, 1600] loss: 1.087', '[2, 1800] loss: 1.118', '[2, 2000] loss: 1.104', '[2, 2200] loss: 1.078', '[2, 2400] loss: 1.098', '[2, 2600] loss: 1.109', '[2, 2800] loss: 1.089', and '[2, 3000] loss: 1.068'.

```
Every 1.0s: tail job-11121.out

[2, 1200] loss: 1.101
[2, 1400] loss: 1.115
[2, 1600] loss: 1.087
[2, 1800] loss: 1.118
[2, 2000] loss: 1.104
[2, 2200] loss: 1.078
[2, 2400] loss: 1.098
[2, 2600] loss: 1.109
[2, 2800] loss: 1.089
[2, 3000] loss: 1.068
```

HUITA

基本模型理解

目的也是想预测item和user的匹配度进行推送

创新点为层次化的attention: Three-Tier Attention for Recommendation

- 用户有很多review，对每条review有不同的attention
- review有很多句子，每条句子有不同的attention
- 句子有很多word，每个单词有不同的attention

关于cnn使用两次的思考

review前后之间没有语义关系，使用cnn提取特征是无效的

word和sent前后之间有语义关系，使用cnn提取特征

对上次实现的改进

完善了word embedding

word-embedding.py 同nrpa

model部分

`huita.py` 同样使用padding - reshape

一开始loss爆炸了，只调节最后一层参数没有用

将其余层的V b也置零

第一轮loss稳定在了1

但是后面还是有问题（还未解决）

padding 确定

review数与nrpa相同

```
----- sent -----
-----sent size-----
100%: 154
98%: 35
98%: 31
95%: 27
93%: 24
90%: 21
88%: 20
85%: 18
----- word -----
-----word size-----
100%: 681
98%: 45
98%: 41
95%: 36
93%: 33
90%: 30
88%: 28
85%: 26
```

最终确定sent_size = 25 word_size=30

数据集

`data.py`

同样是在 `__getitem__` 中加载tensor

最终完成效果

```

[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
----- finish data load -----
----- finish net load -----
----- start train -----
[1, 200] loss: 1.495
[1, 400] loss: 1.153
[1, 600] loss: 1.150
[1, 800] loss: 1.219
[1, 1000] loss: 1.133
[1, 1200] loss: 1.128
[1, 1400] loss: 1.175
[1, 1600] loss: 1.134
[1, 1800] loss: 1.146
[1, 2000] loss: 1.164
[1, 2200] loss: 1.116
[1, 2400] loss: 1.086
[1, 2600] loss: 1.129
[1, 2800] loss: 1.093
[1, 3000] loss: 1.101
[2, 200] loss: 367.790
[2, 400] loss: 1.062
[2, 600] loss: 1.064
[2, 800] loss: 1.101
[2, 1000] loss: 1.131
[2, 1200] loss: 1.134
[2, 1400] loss: 1.131
[2, 1600] loss: 1.140
[2, 1800] loss: 1.166
[2, 2000] loss: 1.115
[2, 2200] loss: 1.105
[2, 2400] loss: 1.101
[2, 2600] loss: 1.100
[2, 2800] loss: 1.101
[2, 3000] loss: 1.098
[3, 200] loss: 339.845
[3, 400] loss: 1.129
[3, 600] loss: 1.075
[3, 800] loss: 1.131
[3, 1000] loss: 1.105
[3, 1200] loss: 1.074
[3, 1400] loss: 1.079
[3, 1600] loss: 1.133
[3, 1800] loss: 1.105
[3, 2000] loss: 1.045

```

实现中遇到的问题

已解决

reshape矩阵

调试中有矩阵尺寸不匹配的情况

word-embedding加载很慢

一开始，每次启动都使用gensim加载embedding weight矩阵

后来将某一次加载的矩阵使用torch.save保存了下来进行复用

gpu占用

训练时遇到CUDA error

发现学校container分派的gpu显存已经被占满

通过stack cat view等方法解决了

未解决

voclist.txt

单词不全

将标点划分出来和某些标点连在一起划分都试过，都有不存在于wordlist的单词

目前的解决方法是一旦产生不知道的单词就置为padding

使用jupyter

训练过程中如果使用jupyter保存每一步运行状态应该会节省很多时间

学校container有相关使用方法

没来得及看

训练中的loss

在每个epoch的第一个minibatch 会出现loss的上升

nrpa比较稳定, 但huita浮动比较大

但是huita loss在第三个epoch比第二个epoch要小

原因可能是初始化参数为0导致的，参数稍增加会导致输出的大幅度改变

可以调节learning rate试一下？