7月16日

一、论文笔记

1. Joint Deep Modeling of Users and Items Using Reviews for Recommendation

1701.04783.pdf (arxiv.org)

1 概述

作者提出了一个 Deep Cooperative Neural Networks（DeepCoNN）的模型，**该模型由耦合在最后一层的两个并行神经网络组成**，其中一个神经网络使用用户撰写的评论对用户的行为进行建模，另一个神经网络使用物品得到的评论信息对物品的特性进行建模，最后一层将两个并行神经网络分别得到的x_u和y_i拼接成一个向量z，最后利用Factorization Machine (FM)来捕捉用户行为和物品属性的特征以及他们之间的成对交互联系。

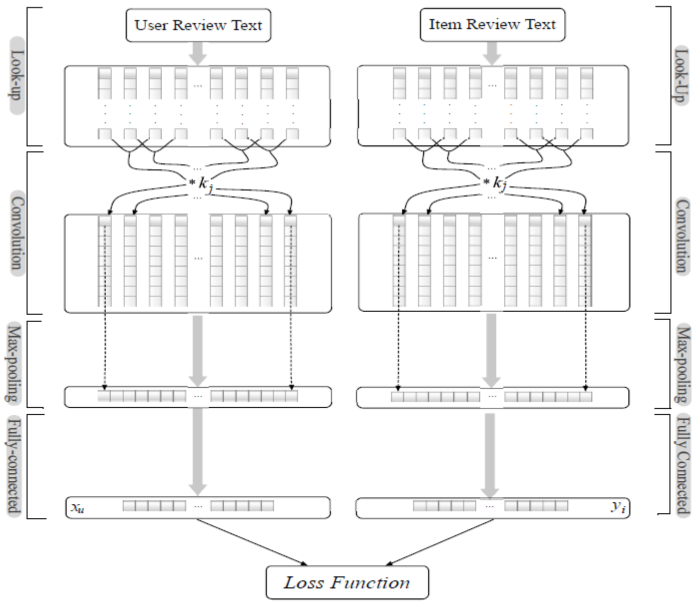


Figure 1: The architecture of the proposed model

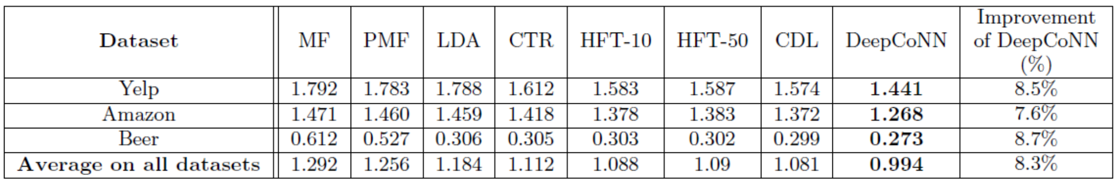
2 创新点

- DeepCoNN 是第一个使用神经网络从评论中联合建模用户和项目的网络。
- DeepCoNN 使用预训练的词嵌入技术从评论中提取语义信息来表示评论文本，而非其他论文通常会使用的bag-of-words技术，词嵌入技术保留并考虑了文本中词语的顺序信息。
- DeepCoNN 可以根据训练数据的大小进行扩展，并且由于它基于 NN，因此可以轻松地使用新数据进行训练和更新。

3 实验结果

- 在多个数据集上平均8.3%的准确率提升。

Table 3: MSE Comparison with baselines. Best results are indicated in bold.



Dataset	MF	PMF	LDA	CTR	HFT-10	HFT-50	CDL	DeepCoNN	Improvement of DeepCoNN (%)
Yelp	1.792	1.783	1.788	1.612	1.583	1.587	1.574	**1.441**	8.5%
Amazon	1.471	1.460	1.459	1.418	1.378	1.383	1.372	**1.268**	7.6%
Beer	0.612	0.527	0.306	0.305	0.303	0.302	0.299	**0.273**	8.7%
Average on all datasets	1.292	1.256	1.184	1.112	1.088	1.09	1.081	**0.994**	8.3%

- 实验结果表明，对于评论或评分很少的用户和项目，DeepCoNN 比 MF 获得更多的 MSE 减少。 特别是当只有一个评论可用时，DeepCoNN 获得了最大的 MSE 减少。 因此，它验证了 DeepCoNN 可以有效地缓解数据稀疏问题。

4 论文复现

附上代码GitHub链接

4.1 Environments

- python 3.8
- pytorch 1.70

4.2 Dataset

- 首先进行数据集的下载与预处理：评论信息有关的数据集下载链接：

  http://deepyeti.ucsd.edu/jianmo/amazon/categoryFilesSmall/Digital_Music_5.json.gz

- 下载之后放置路径为`data/Digital_Music_5.json.gz`
- 然后运行数据预处理程序对评论信息数据集进行预处理：

```
python preprocess.py
```

- 处理之后产生train.csv，valid.csv 和 test.csv三个文件。
- 下载预先训练好的词向量文件：

  https://nlp.stanford.edu/data/glove.6B.zip

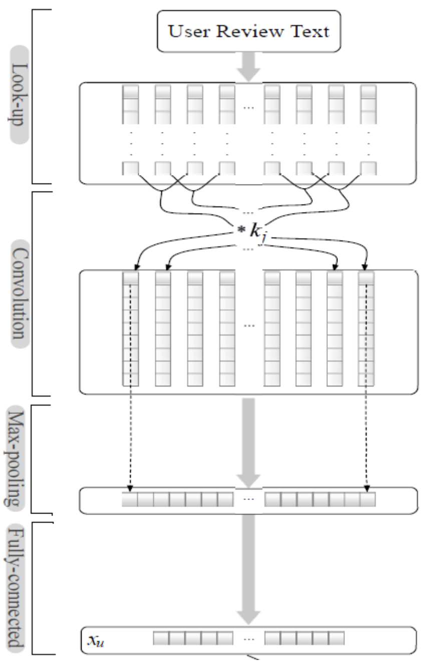
- 下载之后放置路径为`embedding/glove.6B.50d.txt`

4.3 代码解析

```
preprocess.py: 对数据集进行预处理的代码，需要提前单独运行
    config.py: 对各种网络参数的调整以及读取文件的路径都在此文件中进行
     model.py: 定义了DeepCoNN的网络结构
     utils.py: 过程中需要用到的一些工具函数
      main.py: 包含训练和测试过程的主函数
```

着重来看一下model.py中的内容，model的定义分为三个部分：

- class CNN(nn.Module)

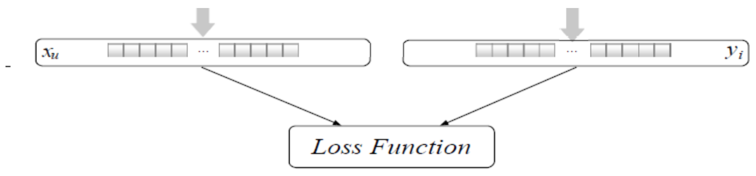


```python
# 定义了分别处理用户评论和物品评论的单个CNN的结构
class CNN(nn.Module):

    def __init__(self, config, word_dim):
        super(CNN, self).__init__()

        self.kernel_count = config.kernel_count
        self.review_count = config.review_count

        # self.conv为卷积层，也就是上图中的上面三层
        self.conv = nn.Sequential(
            # 使用卷积核对输入评论的词向量表示进行卷积操作
            # 输出形状：(new_batch_size, kernel_count, review_length)
            nn.Conv1d(
                in_channels=word_dim,
                out_channels=config.kernel_count,
                kernel_size=config.kernel_size,
                padding=(config.kernel_size - 1) // 2),

            # 激活函数为ReLU()
            nn.ReLU(),

            # 通过MaxPool层，max pooling的窗口形状为(1,
config.review_length)
            # 输出形状：(new_batch_size,kernel_count,1)
            nn.MaxPool2d(kernel_size=(1, config.review_length)),

            # 最后通过Dropout层，减小overfitting的出现几率
            nn.Dropout(p=config.dropout_prob))

        # self.linear为全连接层，也就是上图中的最后一层
        self.linear = nn.Sequential(
            # 全连接层，该全连接层的神经元个数为cnn_out_dim
            nn.Linear(config.kernel_count * config.review_count, config.cnn_out_dim),

            # 激活函数为ReLU()
            nn.ReLU(),

            # 最后通过Dropout层，减小overfitting的出现几率
            nn.Dropout(p=config.dropout_prob))

    # 输入形状：(new_batch_size, review_length, word2vec_dim)
    def forward(self, vec):
        # out(new_batch_size, kernel_count, 1) kernel count指一条评论潜在向量
        latent = self.conv(vec.permute(0, 2, 1))
        latent = self.linear(latent.reshape(-1, self.kernel_count * self.review_count))
        # 最终输出形状：(batch_size, cnn_out_dim)
        return latent
```

- class FactorizationMachine(nn.Module)

  推荐系统FM & FFM算法解读与实践_baymax_007的博客-CSDN博客

  推荐系统召回四模型之：全能的FM模型 - 知乎 (zhihu.com)



```python
# 定义了捕捉用户行为和物品属性的特征以及他们之间的成对交互联系的FM层
class FactorizationMachine(nn.Module):
def __init__(self, p, k):  # p=cnn_out_dim
    super(FactorizationMachine, self).__init__()
    # 通过设置nn.Parameter，使得self.v成为模型的一个参数，在训练过程中能够更新
    self.v = nn.Parameter(torch.zeros(p, k))
    self.linear = nn.Linear(p, 1, bias=True)

def forward(self, x):
    linear_part = self.linear(x)  # input shape(batch_size, cnn_out_dim), out shape(batch_size, 1)
    inter_part1 = torch.mm(x, self.v)
    inter_part2 = torch.mm(x ** 2, self.v ** 2)
    pair_interactions = torch.sum(inter_part1 ** 2 - inter_part2, dim=1)
    output = linear_part.transpose(1, 0) + 0.5 * pair_interactions
    return output.view(-1, 1)  # out shape(batch_size, 1)
```

- class DeepCoNN(nn.Module)

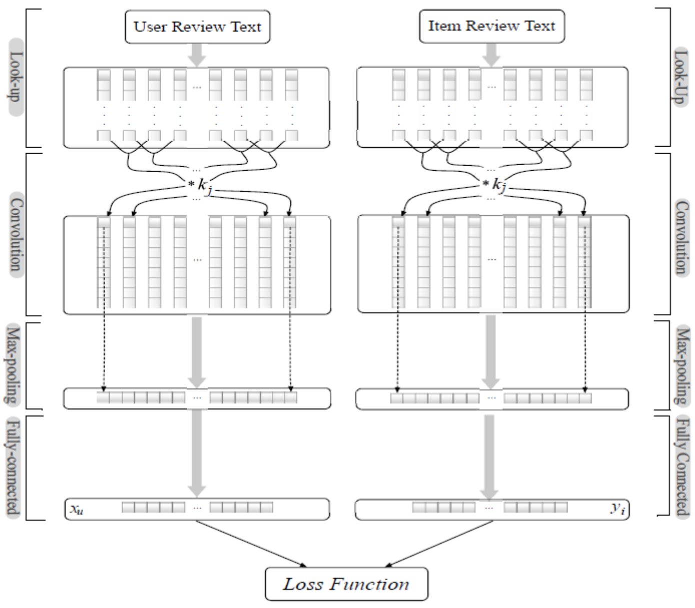


Figure 1: The architecture of the proposed model

```python
class DeepCoNN(nn.Module):
def __init__(self, config, word_emb):
    super(DeepCoNN, self).__init__()
    self.embedding = 
nn.Embedding.from_pretrained(torch.Tensor(word_emb))
    # 定义了处理user review的CNN
    self.cnn_u = CNN(config, word_dim=self.embedding.embedding_dim)
    # 定义了处理item review的CNN
    self.cnn_i = CNN(config, word_dim=self.embedding.embedding_dim)
    # 定义了最后的FM层
    self.fm = FactorizationMachine(config.cnn_out_dim * 2, 10)

def forward(self, user_review, item_review):  # input shape(batch_size, 
review_count, review_length)
    new_batch_size = user_review.shape[0] * user_review.shape[1]
    user_review = user_review.reshape(new_batch_size, -1)
    item_review = item_review.reshape(new_batch_size, -1)

    # 用户和物品评论转换为词向量表达
    u_vec = self.embedding(user_review)
    i_vec = self.embedding(item_review)

    # 用户和物品评论的词向量分别输入两个CNN
    user_latent = self.cnn_u(u_vec)
    item_latent = self.cnn_i(i_vec)

    # 两个CNN的输出拼接在一起，然后输给FM，FM的输出即为预测结果
    concat_latent = torch.cat((user_latent, item_latent), dim=1)
    prediction = self.fm(concat_latent)
    return prediction
```

4.4 实际训练结果

```
2021-07-16T03:13:56.737996451Z SYSTEM: Preparing env...
2021-07-16T03:13:57.341554983Z SYSTEM: Running...
2021-07-16T03:13:58.705346378Z PAD_WORD = <UNK>
2021-07-16T03:13:58.705381858Z batch_size = 128
2021-07-16T03:13:58.705392987Z cnn_out_dim = 50
2021-07-16T03:13:58.705401758Z device = cuda:0
2021-07-16T03:13:58.705410353Z dropout_prob = 0.5
2021-07-16T03:13:58.705489453Z kernel_count = 100
2021-07-16T03:13:58.705519542Z kernel_size = 3
2021-07-16T03:13:58.70552887Z l2_regularization = 1e-06
2021-07-16T03:13:58.705537258Z learning_rate = 0.002
2021-07-16T03:13:58.705544951Z learning_rate_decay = 0.99
2021-07-16T03:13:58.705555378Z lowest_review_count = 2
2021-07-16T03:13:58.705563165Z model_file = model/best_model.pt
2021-07-16T03:13:58.705570838Z review_count = 10
2021-07-16T03:13:58.705578113Z review_length = 40
2021-07-16T03:13:58.705586719Z test_file = data/music/test.csv
2021-07-16T03:13:58.705594402Z train_epochs = 20
2021-07-16T03:13:58.705672167Z train_file = data/music/train.csv
2021-07-16T03:13:58.705684976Z valid_file = data/music/valid.csv
2021-07-16T03:13:58.70569353Z word2vec_file = embedding/glove.6B.50d.txt
2021-07-16T03:13:58.705731934Z
2021-07-16T03:13:58.705795721Z 2021-07-16 11:13:58## Load embedding and data...
2021-07-16T03:15:36.028614578Z 2021-07-16 11:15:36## Start the training!
2021-07-16T03:15:40.096962657Z 2021-07-16 11:15:40#### Initial train mse 23.341680, validation mse 22.699222
2021-07-16T03:15:48.123525317Z 2021-07-16 11:15:48#### Epoch   0; train mse 1.038468; validation mse 0.654934
2021-07-16T03:15:57.574551143Z 2021-07-16 11:15:57#### Epoch   1; train mse 0.728854; validation mse 0.552114
2021-07-16T03:16:06.540706996Z 2021-07-16 11:16:06#### Epoch   2; train mse 0.593755; validation mse 0.505486
2021-07-16T03:16:15.623477204Z 2021-07-16 11:16:15#### Epoch   3; train mse 0.505096; validation mse 0.515194
2021-07-16T03:16:23.818348699Z 2021-07-16 11:16:23#### Epoch   4; train mse 0.461765; validation mse 0.501074
2021-07-16T03:16:33.201231077Z 2021-07-16 11:16:33#### Epoch   5; train mse 0.439223; validation mse 0.499264
2021-07-16T03:17:07.990007044Z 2021-07-16 11:17:07#### Epoch   6; train mse 0.431829; validation mse 0.495032
2021-07-16T03:17:16.990821311Z 2021-07-16 11:17:16#### Epoch   7; train mse 0.429399; validation mse 0.501746
2021-07-16T03:17:24.961099926Z 2021-07-16 11:17:24#### Epoch   8; train mse 0.423754; validation mse 0.492980
2021-07-16T03:17:34.099448969Z 2021-07-16 11:17:34#### Epoch   9; train mse 0.410848; validation mse 0.498421
2021-07-16T03:17:42.107455596Z 2021-07-16 11:17:42#### Epoch  10; train mse 0.408566; validation mse 0.488360
2021-07-16T03:17:51.027413967Z 2021-07-16 11:17:51#### Epoch  11; train mse 0.407218; validation mse 0.495572
2021-07-16T03:17:59.018639745Z 2021-07-16 11:17:59#### Epoch  12; train mse 0.404888; validation mse 0.497951
2021-07-16T03:18:07.024345422Z 2021-07-16 11:18:07#### Epoch  13; train mse 0.401306; validation mse 0.487931
```

```
2021-07-16T03:18:15.978997684Z 2021-07-16 11:18:15#### Epoch  14; train mse
0.398702; validation mse 0.502684
2021-07-16T03:18:24.007130256Z 2021-07-16 11:18:24#### Epoch  15; train mse
0.388489; validation mse 0.486976
2021-07-16T03:18:33.304226436Z 2021-07-16 11:18:33#### Epoch  16; train mse
0.385698; validation mse 0.501626
2021-07-16T03:18:41.263538644Z 2021-07-16 11:18:41#### Epoch  17; train mse
0.382398; validation mse 0.487572
2021-07-16T03:18:49.207497044Z 2021-07-16 11:18:49#### Epoch  18; train mse
0.379623; validation mse 0.476653
2021-07-16T03:18:58.195607755Z 2021-07-16 11:18:58#### Epoch  19; train mse
0.378280; validation mse 0.492297
2021-07-16T03:18:58.1956591Z 2021-07-16 11:18:58## End of training! Time
used 198 seconds.
2021-07-16T03:18:58.379652888Z 2021-07-16 11:18:58## Start the testing!
2021-07-16T03:18:58.485499736Z 2021-07-16 11:18:58## Test end, test mse is
0.456900, time used 0 seconds.
2021-07-16T03:18:59.533740185Z SYSTEM: Finishing...
2021-07-16T03:19:14.406839182Z SYSTEM: Done!
```

可以看到最终在测试集上的mse 为 0.456900，使用DeepCoNN确实得到了不错的结果。