

深度学习导论大作业报告

作者：林威 19351024

深度学习导论大作业报告

- 一、背景介绍
- 二、数据集介绍
- 三、研究回顾
- 四、模型结构
- 五、实验与结果
 - (1) 矩阵分解模型
 - (2) 深度学习模型
 - (3) 改进后的深度学习模型
- 六、总结与未来方向
- 附录
 - main.py
 - model.py
 - Train.py
 - Plot.py

一、背景介绍

随着多年来网络上的数据不断增加，从电子商务到电子资源，推荐系统已经进入了人们的视野。如今，Netflix、Amazon、YouTube等巨头使用推荐系统来帮助用户找到有用的信息，从而提高用户的体验，从而获得用户的信心。为了从网络上的海量数据中为用户提供相关信息，同时帮助公司盈利，推荐系统会向用户推荐一系列有趣且相关的项目。换句话说，推荐系统是一种工具，它可以预测用户在不久的将来可能会消费的商品。

在关于推荐系统的研究的时间线中，有一个不得不提的时间点，那就是2006年的10月2日。在这一天Netflix启动了Netflix Prize Competition¹。这是一个机器学习和数据挖掘的比赛，旨在解决电影评分预测问题。Netflix举办这个比赛的目的是为了发现更好的方法来向用户推荐产品，这是他们商业模式的核心任务。第一个将Netflix的Cinematch系统的准确率提升10%的团队将获胜。他们会获得一百万美元的大奖。

在本次大作业中，我将会聚焦于这个问题，尝试用深度学习的方法来获得较好的结果。

二、数据集介绍

Netflix提供了480,189名用户对17,770部电影进行的100,480,507次评分的训练数据集。每个训练数据是一个长度为4的元组，分别表示了用户、电影、评分日期与评分的信息。用户和电影是用整数来表示的，而评分则是从1到5的整数。在这个数据集当中，每个用户平均对超过200部电影进行了评分，每部电影平均被超过5000名用户打分。但是不同用户或者电影之间存在很大的差异，一些电影只有3个评分²，最多的一个用户对超过17000部电影进行了评分³。

在这里我对问题进行了一个简化，时间因素将不被考虑。因此有关评分日期的信息将不会作为推荐系统的输入。另外，由于我的笔记本电脑内存有限，我并没有选取Netflix提供的所有数据，而是选取了前10,000名用户（根据用户ID排序）的评分信息。在本次大作业中所用到的数据集可以用下表来表示。

	Movie1	Movie2	Movie3	Movie4	...	Movie17,770
User1	5	3	?	?	...	?
User2	4	?	?	?	...	1
User3	?	?	4	?	...	?
User4	?	2	?	?	...	?
...
User10,000	?	?	?	3	...	?

在这个被选取的数据集中，一共有100,480,507个评分信息。可以看到，这是一个十分稀疏的矩阵，它的密度只有0.117，而我的目标就是要填补那些空白的地方。

一个值得注意的点是，选取更小的数据集可能会加大电影评分预测问题的难度。因为对于每个用户（电影）来说，他会有更少的“邻居”，即相似的用户（电影），这样我在预测该用户（电影）的评分时，受个别极端（噪音）评分的影响可能会更大，不利于预测的结果。

三、研究回顾

推荐系统通常被分为协同过滤（collaborative filtering）和基于内容的过滤（content-based filtering）这两大类。

内容过滤方法会对每个用户或者产品创建一个“档案”。举个例子，对于电影来说，它的“档案”即包括了它的流派、出演演员、导演以及票房等相关信息。用户档案则可能包括一些个人背景特征和对调查问卷的回答。程序将根据这些“档案”将用户与对应的产品联系起来。当然，基于内容的策略需要收集很多额外的信息，这在实际应用中可能会花费很多人力物力。一个比较著名的例子是Music Genome Project⁴。在这个计划当中，音乐被用数学的方式来量化，以用于更好的匹配音乐与对应的听众。

协同过滤只依赖于过去的用户行为，例如以前的交易或产品评级，而不需要创建显式的“档案”。协同过滤分析用户之间的关系和产品之间的相互依赖关系，以识别新的用户-项目关联。协作过滤的一个强大之处在于它并不需要领域知识。它可以解决数据方面的问题，这些数据方面往往难以捉摸，难以使用内容过滤进行剖析。虽然协同过滤通常比基于内容的技术更准确，但由于无法解决系统的新产品和用户的问题，它遭遇了所谓的“冷启动”问题。在这方面，内容过滤更为优越。协同过滤的两个主要领域是邻域方法（Neighborhood methods）和潜在因素模型（Latent factor models）。邻域方法集中于计算物品之间或者用户之间的关系。基于物品的邻域方法会根据同一用户对“邻近”物品的评价来评估用户对某物品的偏好。潜在因素模型是另一种预测评分的方法，它通过从现有评分推断出的若干个潜在因素来描述商品和用户。

矩阵分解（Matrix Factorization）是近年来非常流行的潜在因素模型⁵。矩阵因子分解的基本形式是通过从物品评级模式中推断出的因子向量来表征物品和用户。项目和用户因素之间的高度对应导致推荐。这种方法结合了良好的可扩展性和预测准确性，它们也为建模各种现实情况提供了很大的灵活性。

具体来说，矩阵分解模型将用户和物品映射到一个维度为 f 的联合潜在因素空间，这样用户与物品的交互被建模为该空间中的内积。相对应的，每个物品 i 被表示为一个向量 $\vec{q}_i \in R^f$ ，每个用户 u 也被表示成一个向量 $\vec{p}_u \in R^f$ 。对于一个给定物品 i ， q_i 的各个分量表示该物品在对应的潜在因素上的表现。对于一个给定用户 u ， p_u 的各个分量表示该用户对于对应的潜在因素感兴趣的程度。它们的内积， $q_i^T p_u$ 则被认为可以表现用户与物品之间的关系。因此该模型预测的用户评分可以用下式来表示：

$$\hat{r}_{ui} = q_i^T p_u \quad q_i, p_u \in R^f$$

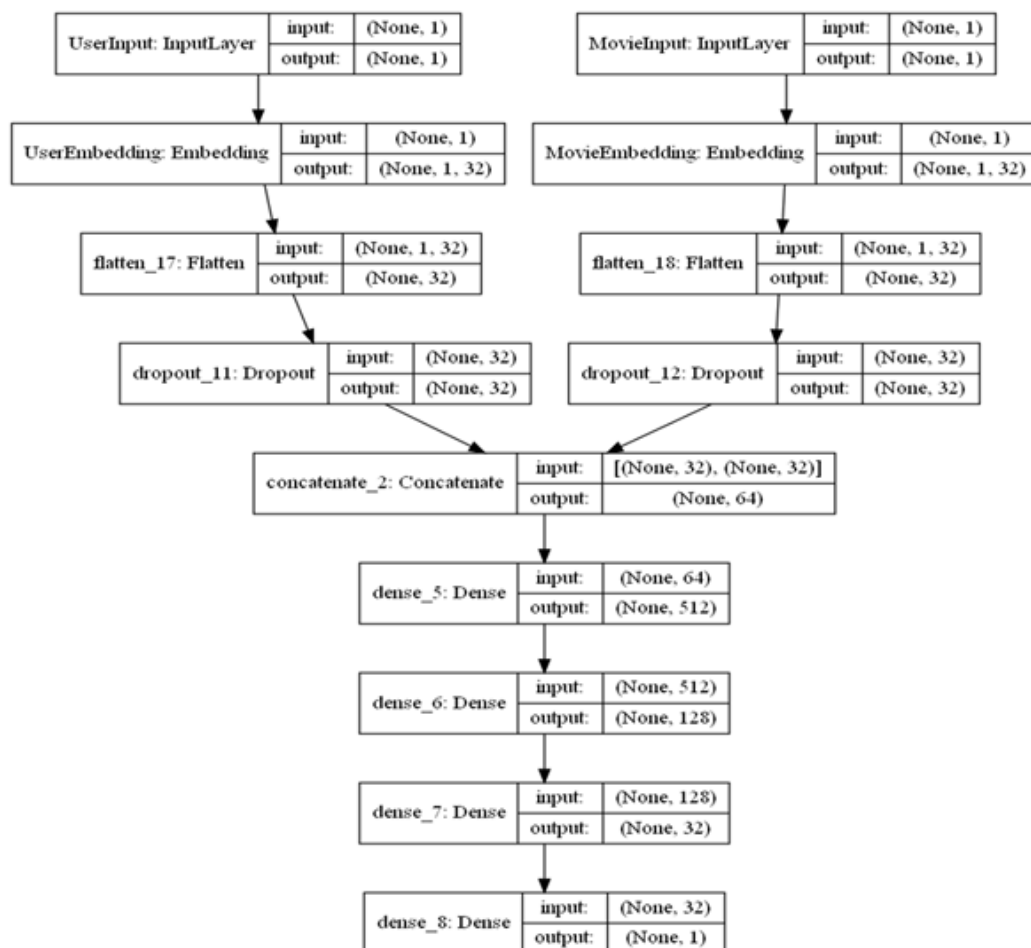
模型的误差用正规化的平方误差（regularized squared error）来衡量：

$$\min_{q^*, p^*} \sum_{(u,i) \in k} (r_{ui} - q_i^T p_u)^2 + \lambda(|q_i|^2 + |p_u|^2)$$

其中， k 是所有评分已知的 (u, i) 的集合，即训练集。

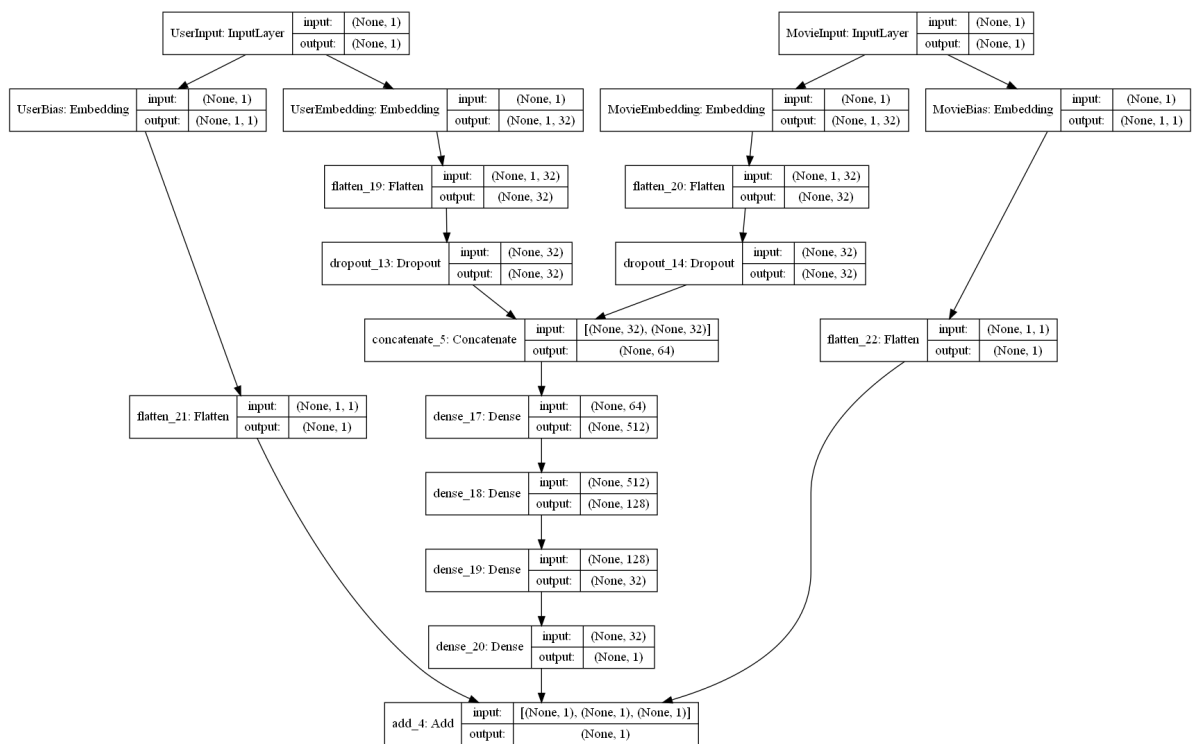
四、模型结构

在经典的矩阵分解模型之中，我们用向量的内积来表示预测的结果。一个自然的想法是将这个简单的内积操作替换成一个神经网络。通过引入深度学习的内容来预测电影评分。所得到的模型结构如下图所示，其一共有782,529个参数。



该模型一共有两个输入，分别是UserID和MovieID。这两者分别会先过一个embedding layer，得到其对应的潜在因素向量。然后将这两个向量接合在一起，再通过一个具有三个隐含层的全链接神经网络来预测最终的评分。

这是一个最初步的设想，再考虑更多因素后，我又对模型进行了改进。考虑到每个用户的打分标准不一样，每个电影的评分之间也有差异，我对最终的评分预测进行了一个修正，加入了User bias和Movie bias两项。改进后的模型如下图所示，一共有803,761个参数。



五、实验与结果

在本节当中，我在选取的数据集上对经典矩阵分解模型、简易DL模型和改进后的DL模型进行了测试。所有实验均在我自己的笔记本电脑上完成，并使用了GPU加速。电脑配置为如下表所示：

处理器	RAM	GPU
Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz 1.99 GHz	8GB	MX250

我将所有的评分信息以9：1的比例分为训练集和测试集，用于后续的实验。

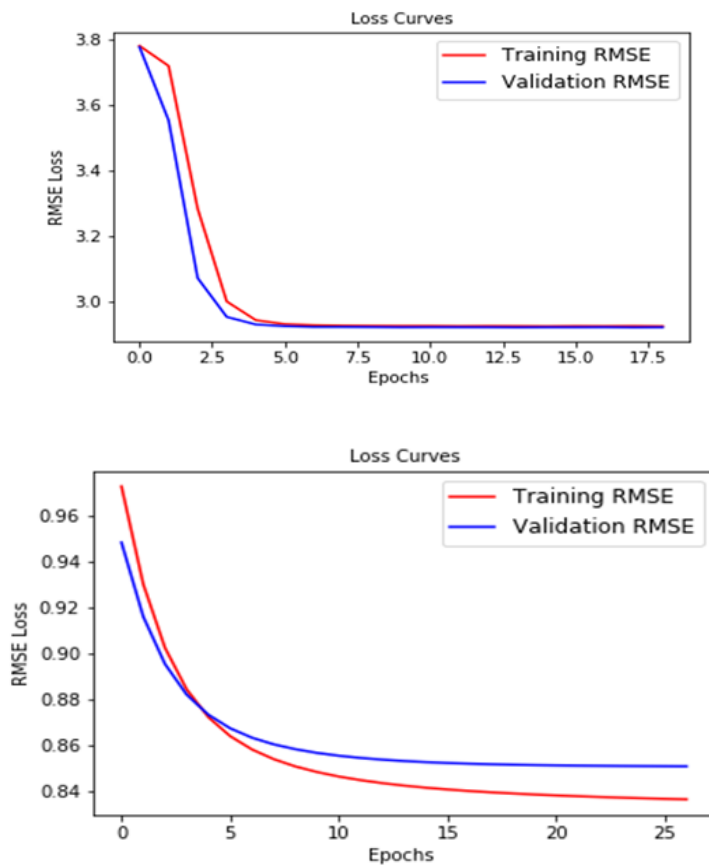
(1) 矩阵分解模型

为了衡量DL模型的表现，我先复现了矩阵分解模型，并在数据集上测试了该模型的表现，一些具体的参数和结果如下：

- latent factor: 32
- Batch size: 4096
- drop out: 0.3
- Training time: 4s/epoch × 50epochs
- 在训练集和测试集的结果如下

	Training RMSE	Testing RMSE
MF	2.915	2.923
MF with normalization	0.912	0.931

下图（左）为MF的结果，由loss curve可以得知模型大约在第5个epoch就已经收敛，RMSE约为2.9左右。下图（右）为做Normalization之后MF的结果。注意图中的RMSE Loss并不是真正的结果，而是经过Normalization之后的数据的RMSE，较真正结果相比会偏小。经过Normalization之后的结果有了很大的改善，最小能到0.9左右，收敛时间也大约是在10多个epoch左右。



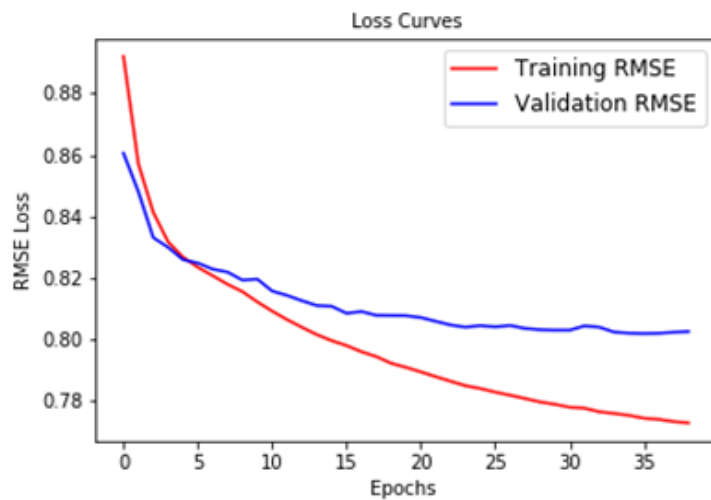
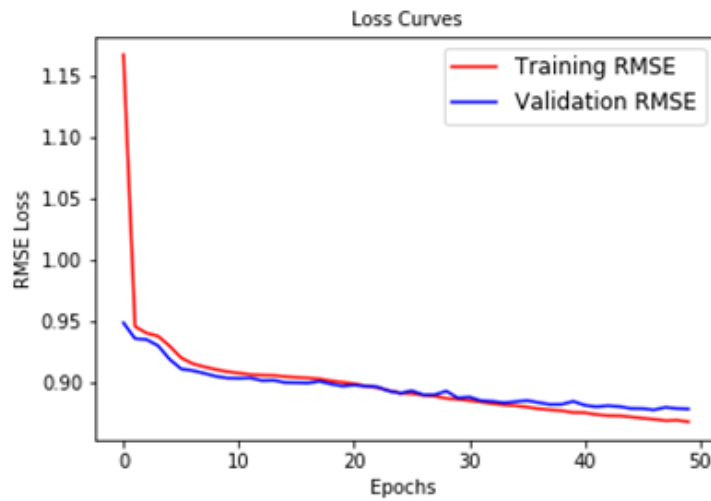
(2) 深度学习模型

我接着对深度学习模型也做了同样的测试，参数与结果如下：

- latent factor: 32
- Batch size: 4096
- drop out: 0.3
- Training Time: 10s/epoch \times 50epochs
- 在训练集和测试集的结果如下

	Training RMSE	Testing RMSE
DL	0.8235	0.878
DL with normalizaiton	0.831	0.882

下图（左）为DL模型的结果，可以看到相对于MF模型来说DL模型有了很大的改进。下图（右）为Normalization之后DL的表现。同样的，图中的RMSE并不是真正的结果。实际上，对于简单的DL模型来说，normalization并没有起到很好的效果。



另外我还测试了不同latent factor对DL模型的影响，结果如下：

Latent factor	32	64	128	256	512
Training RMSE	0.819	0.817	0.816	0.821	0.842
Test RMSE	0.874	0.873	0.874	0.874	0.883

可以看到实际上Latent factor对结果并没有很大的影响，但越小的latent factor运行时间越短。因此我这里选择了latent factor=32。

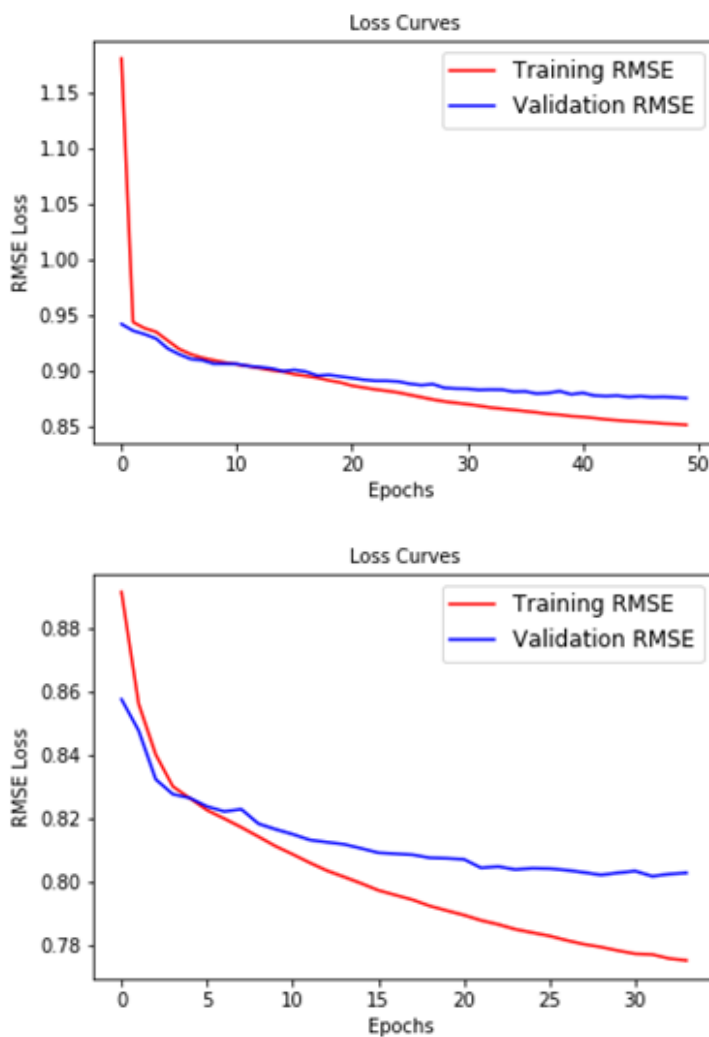
(3) 改进后的深度学习模型

通过引入User bias和Movie bias，可以得到改进后的深度学习模型。其参数和结果如下：

- latent factor: 32
- Batch size: 4096
- drop out: 0.3
- Training Time: 11s/epoch × 50epochs
- 在训练集和测试集的结果如下

	Training RMSE	Testing RMSE
DL+	0.817	0.873
DL+ with normalization	0.806	0.871

下图（左）为改进后的DL模型的结果，可以看到相对于本来的DL模型有了一定的。下图（右）为Normalization之后改进的DL模型的表现。同样的，图中的RMSE并不是真正的结果。但在改进之后的DL模型，Normalization是有积极作用的。



最后将我最终的结果与Netflix比赛中的几个baseline进行比较：

Netflix	10% Improvement	2007 Progress Prize	2008 Progress Prize	2009 Grand Prize	My Result
0.9514	0.8563	0.8712	0.8616	0.8554	0.871

可以看到这个简单的DL模型已经可以超过2007年的Progress Prize（\$50000奖金）的成绩了，而比上Netflix最开始的结果更是要好上不少。但相比较于最终获大奖的算法来说，我的DL模型仍然具有很大的发展空间。

六、总结与未来方向

通过这门课的学习，我了解到了深度学习最基本的一些知识。在这次大作业中，将这些知识运用到了一个简单的例子上，也获得了不错的结果，可见深度学习的强大之处。

在未来，可能可以在以下几个方面对我的模型进行改进：

- 引入时间数据，考虑用户随时间变换的兴趣转移
- 进一步构造更复杂的模型结构，而并非简单的全链接网络
- 考虑Ensemble model，组合多个模型来优化结果

附录

本次大作业的所有代码包括四个py文件，其各自用途如下：

- main.py 运行实验的主程序
- model.py 包含MF与DL的模型，以及一些工具类函数
- Train.py 用于训练模型
- Plot.py 用于画出模型图与loss curve

以下是各个文件的代码

main.py

```
1 import os
2 import numpy as np
3 import pandas as pd
4 import Train, Model, Plot
5
6
7 def main(boolNormalize, boolDeep, boolBias, intLatentSize=32):
8     strProjectFolder = os.path.dirname(__file__)
9     strOutputPath = Model.get_path(boolNormalize, boolDeep,
10 boolBias, intLatentSize)
11
12     DataTrain = pd.read_csv(os.path.join(strProjectFolder,
13     "../data/netflixData/rating.csv"), usecols=["userId_nor", "movieId_nor",
14     "rating"])
15
16     DataTrain = DataTrain.sample(frac=1, random_state=10)
17     intUserSize = max(DataTrain['userId_nor']) + 1
18     intMovieSize = max(DataTrain['movieId_nor']) + 1
19
20     arrayUsers = DataTrain["userId_nor"].values
21     arrayMovies = DataTrain["movieId_nor"].values
22     arrayRate = DataTrain["rating"].values
23
24     intVaildSize = int(len(DataTrain) * 0.1)
25     arrayTrainUser = arrayUsers[:-intVaildSize]
26     arrayTrainMovie = arrayMovies[:-intVaildSize]
27     arrayTrainRate = arrayRate[:-intVaildSize]
28
29     arrayValidUser = arrayUsers[-intVaildSize:]
30     arrayValidMovie = arrayMovies[-intVaildSize:]
31     arrayValidRate = arrayRate[-intVaildSize:]
32
33     arrayTrainRateAvg = np.mean(arrayTrainRate)
34     arrayTrainRateStd = np.std(arrayTrainRate)
```



```

32     #std=1.0860
33     if boolNormalize:
34         arrayTrainRate = (arrayTrainRate -
arrayTrainRateAvg)/arrayTrainRateStd
35         arrayValidRate = (arrayValidRate -
arrayTrainRateAvg)/arrayTrainRateStd
36
37         Train.getTrain(arrayTrainUser=arrayTrainUser,
arrayTrainMovie=arrayTrainMovie, arrayTrainRate=arrayTrainRate
38             , arrayValidUser=arrayValidUser,
arrayValidMovie=arrayValidMovie, arrayValidRate=arrayValidRate
39             , intUserSize=intUserSize
40             , intMovieSize=intMovieSize
41             , intLatentSize=intLatentSize
42             , boolBias=boolBias
43             , boolDeep=boolDeep
44             , strProjectFolder=strProjectFolder,
strOutputPath=strOutputPath)
45
46
47         arrayTrainPredict = Model.makePredict(arrayTrainUser, arrayTrainMovie,
strProjectFolder=strProjectFolder, strOutputPath=strOutputPath)
48
49         if boolNormalize:
50             arrayTrainPredict = (arrayTrainPredict * arrayTrainRateStd) +
arrayTrainRateAvg
51             trainMSE = np.mean(np.square(np.squeeze(arrayTrainPredict) -
arrayRate[: -intVaildSize]))
52             trainRMSE = np.sqrt(np.mean(np.square(np.squeeze(arrayTrainPredict) -
arrayRate[: -intVaildSize])))
53
54             arrayValidPredict = Model.makePredict(arrayValidUser, arrayValidMovie,
strProjectFolder=strProjectFolder, strOutputPath=strOutputPath)
55             if boolNormalize:
56                 arrayValidPredict = (arrayValidPredict * arrayTrainRateStd) +
arrayTrainRateAvg
57                 validMSE = np.mean(np.square(np.squeeze(arrayValidPredict) - arrayRate[-
intVaildSize:]))
58                 validRMSE = np.sqrt(np.mean(np.square(np.squeeze(arrayValidPredict) -
arrayRate[-intVaildSize:])))
59 #
60         print("latentSize:{} TrainRMSE:{:.4f} ValidRMSE:
{:.4f}".format(intLatentSize,trainRMSE,validRMSE))
61 #     print(intLatentSize, trainRMSE, validRMSE)
62     Plot.plotLossAccuracyCurves(strOutputPath)
63     Plot.plot_model(strOutputPath)
64     return intLatentSize,trainMSE, trainRMSE, validMSE, validRMSE
65
66 #     Plot.plotModel(strOutputPath)
67
68 if __name__ == "__main__":
69 #     la_size=[32,64,128,256,512]
70     la_size=[32]
71     results=[]
72     for la in la_size:
73         re=main(1,1,1,la)
74         results.append(re)

```

model.py

```
1  from keras.models import Sequential, Model
2  from keras.regularizers import l2
3  from keras.layers import Input, Embedding, Flatten, Dot, Add, Dropout,
   Concatenate, Dense
4  from keras.optimizers import Adam
5  import keras.backend as K
6  from keras.models import load_model
7  import os
8  import numpy as np
9
10
11 def MF(intUsersize, intMovieSize, intLatentSize, boolBias):
12     UserInput = Input(shape=(1, ), name="UserInput")
13     UserEmbedding = Embedding(intUsersize, intLatentSize,
   embeddings_initializer="random_normal", embeddings_regularizer=l2(0.001),
   name="UserEmbedding")(UserInput)
14     UserEmbedding = Flatten()(UserEmbedding)
15     UserEmbedding = Dropout(0.3)(UserEmbedding)
16
17     MovieInput = Input(shape=(1, ), name="MovieInput")
18     MovieEmbedding = Embedding(intMovieSize, intLatentSize,
   embeddings_initializer="random_normal", embeddings_regularizer=l2(0.001),
   name="MovieEmbedding")(MovieInput)
19     MovieEmbedding = Flatten()(MovieEmbedding)
20     MovieEmbedding = Dropout(0.3)(MovieEmbedding)
21     RatingHat = Dot(axes=1)([UserEmbedding, MovieEmbedding])
22
23     if boolBias:
24         UserBias = Embedding(intUsersize, 1,
   embeddings_initializer="zeros", name="UserBias")(UserInput)
25         UserBias = Flatten()(UserBias)
26
27         MovieBias = Embedding(intMovieSize, 1,
   embeddings_initializer="zeros", name="MovieBias")(MovieInput)
28         MovieBias = Flatten()(MovieBias)
29
30         RatingHat = Add()([RatingHat, UserBias, MovieBias])
31
32     model = Model([UserInput, MovieInput], RatingHat)
33
34     optim = Adam()
35     model.compile(optimizer=optim, loss="mse", metrics=[getRMSE])
36     # model.compile(optimizer=optim, loss="mse")
37     # model.summary()
38     return model
39
40
41 def DeepNN(intUsersize, intMovieSize, intLatentSize, boolBias):
42     UserInput = Input(shape=(1, ), name="UserInput")
43     UserEmbedding = Embedding(intUsersize, intLatentSize,
   embeddings_initializer="random_normal", embeddings_regularizer=l2(0.001),
   name="UserEmbedding")(UserInput)
44     UserEmbedding = Flatten()(UserEmbedding)
45     UserEmbedding = Dropout(0.3)(UserEmbedding)
46
```

```

47     MovieInput = Input(shape=(1, ), name="MovieInput")
48     MovieEmbedding = Embedding(intMovieSize, intLatentSize,
embeddings_initializer="random_normal", embeddings_regularizer=l2(0.001),
name="MovieEmbedding")(MovieInput)
49     MovieEmbedding = Flatten()(MovieEmbedding)
50     MovieEmbedding = Dropout(0.3)(MovieEmbedding)
51
52     hidden = Concatenate()([UserEmbedding, MovieEmbedding])
53     hidden = Dense(512, activation="relu")(hidden)
54     hidden = Dense(128, activation="relu")(hidden)
55     hidden = Dense(32, activation="relu")(hidden)
56
57     # RatingHat = Dense(1, activation="relu")(hidden)
58     RatingHat = Dense(1)(hidden)
59
60     if boolBias:
61
62         UserBias = Embedding(intUserSize, 1,
embeddings_initializer="zeros", name="UserBias")(UserInput)
63         UserBias = Flatten()(UserBias)
64
65         MovieBias = Embedding(intMovieSize, 1,
embeddings_initializer="zeros", name="MovieBias")(MovieInput)
66         MovieBias = Flatten()(MovieBias)
67
68         RatingHat = Add()([RatingHat, UserBias, MovieBias])
69
70
71     model = Model([UserInput, MovieInput], RatingHat)
72
73     optim = Adam()
74     model.compile(optimizer=optim, loss="mse", metrics=[getRMSE])
75     # model.summary()
76     return model
77
78
79 def getRMSE(arrayPredict, arrayTrue):
80     # print(arrayPredict.shape)
81     return K.sqrt(K.mean(K.pow(arrayTrue - arrayPredict, 2)))
82
83
84
85 def makePredict(arrayUser, arrayMovie, strProjectFolder, strOutputPath):
86
87     strModelPath = os.path.join(strProjectFolder, strOutputPath +
"model.h5")
88
89     model = load_model(strModelPath, custom_objects={"getRMSE":getRMSE})
90
91     predictions = model.predict([arrayUser, arrayMovie], batch_size=1024)
92
93     return predictions
94
95
96 def get_path(boolNormalize, boolDeep, boolBias, intLatentSize):
97     if boolDeep:
98         if boolBias:
99             strOutputPath = 'Output/DeepBias'

```

```

100         else:
101             strOutputPath = "Output/DeepUnbias"
102         else:
103             if boolBias:
104                 if boolNormalize:
105                     strOutputPath = 'Output/MFBiasNormal'
106                 else:
107                     strOutputPath = "Output/MFBias"
108             else:
109                 if boolNormalize:
110                     strOutputPath = 'Output/MFUnbiasNormal'
111                 else:
112                     strOutputPath = "Output/MFUnbias"
113         return strOutputPath+"_"+str(intLatentSize)

```

Train.py

```

1  import os
2  import numpy as np
3  import Model, Plot
4  from keras.callbacks import EarlyStopping, ModelCheckpoint, CSVLogger
5
6
7  def getTrain(arrayTrainUser, arrayTrainMovie, arrayTrainRate,
8               arrayValidUser, arrayValidMovie, arrayValidRate, intUserSize, intMovieSize,
9               intLatentSize, boolBias, boolDeep, strProjectFolder, strOutputPath):
10
11     if boolDeep:
12         model = Model.DeepNN(intUserSize=intUserSize,
13                               intMovieSize=intMovieSize, intLatentSize=intLatentSize, boolBias=boolBias)
14     else:
15         model = Model.MF(intUserSize=intUserSize, intMovieSize=intMovieSize,
16                           intLatentSize=intLatentSize, boolBias=boolBias)
17
18     callbacks = [EarlyStopping("val_getRMSE", min_delta=0.0005, patience=5)
19                  , ModelCheckpoint(os.path.join(strProjectFolder,
20 strOutputPath + "model.h5"), save_best_only=True)
21                  , CSVLogger(os.path.join(strProjectFolder, strOutputPath +
22 "log.csv"), separator="," , append=False)]
23
24     model.fit([arrayTrainUser, arrayTrainMovie], arrayTrainRate, epochs=50,
25               batch_size=4096, verbose=1, validation_data=([arrayValidUser,
26 arrayValidMovie], arrayValidRate), callbacks=callbacks)
27
28     model.save(os.path.join(strProjectFolder, strOutputPath + "model.h5"))
29
30

```

Plot.py

```





1  import os
2  import numpy as np
3  import pandas as pd
4  import matplotlib.pyplot as plt
5  from keras.models import load_model
6  from keras.utils import plot_model

```

```

7  from sklearn.manifold import TSNE
8  import Model
9
10 def plotModel(strOutputPath):
11     model = load_model(strOutputPath + "model.h5", custom_objects=
12     {"getRMSE": Model.getRMSE})
13     model.summary()
14     plot_model(model, show_shapes=True, to_file=strOutputPath + "model.png")
15
16 def plotLossAccuracyCurves(strOutputPath):
17     pdLog = pd.read_csv(strOutputPath + "log.csv")
18     fig = plt.figure(figsize=(6, 4))
19     # Loss Curves
20     plt.plot(pdLog["epoch"], pdLog["getRMSE"], "r", linewidth=1.5)
21     plt.plot(pdLog["epoch"], pdLog["val_getRMSE"], "b", linewidth=1.5)
22     plt.legend(["Training RMSE", "Validation RMSE"], fontsize=12)
23     plt.xlabel("Epochs ", fontsize=10)
24     plt.ylabel("RMSE Loss", fontsize=10)
25     plt.title("Loss Curves", fontsize=10)
26     plt.savefig(strOutputPath + "LossCurves")
27
28
29
30 #plotModel(Model.get_path(1,1,1,32))
31 #plotLossAccuracyCurves(Model.get_path(True,False,False))

```

-
1. [The Netflix Prize Competition](#) 
 2. ["Miss Congeniality"](#). *Netflix Prize Forum*. 
 3. ["A single customer that rated 17,000 movies"](#). *Netflix Prize Forum*. 
 4. [Music Genome Project](#). Pandora 
 5. [Matrix factorization techniques for recommender systems](#) 