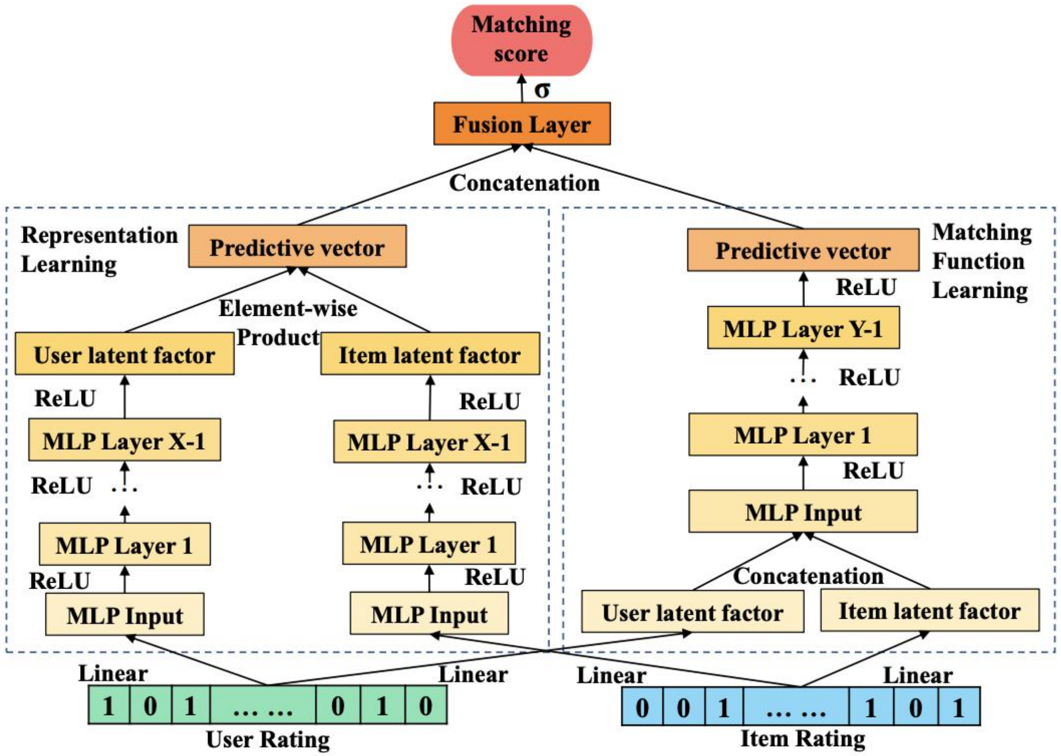
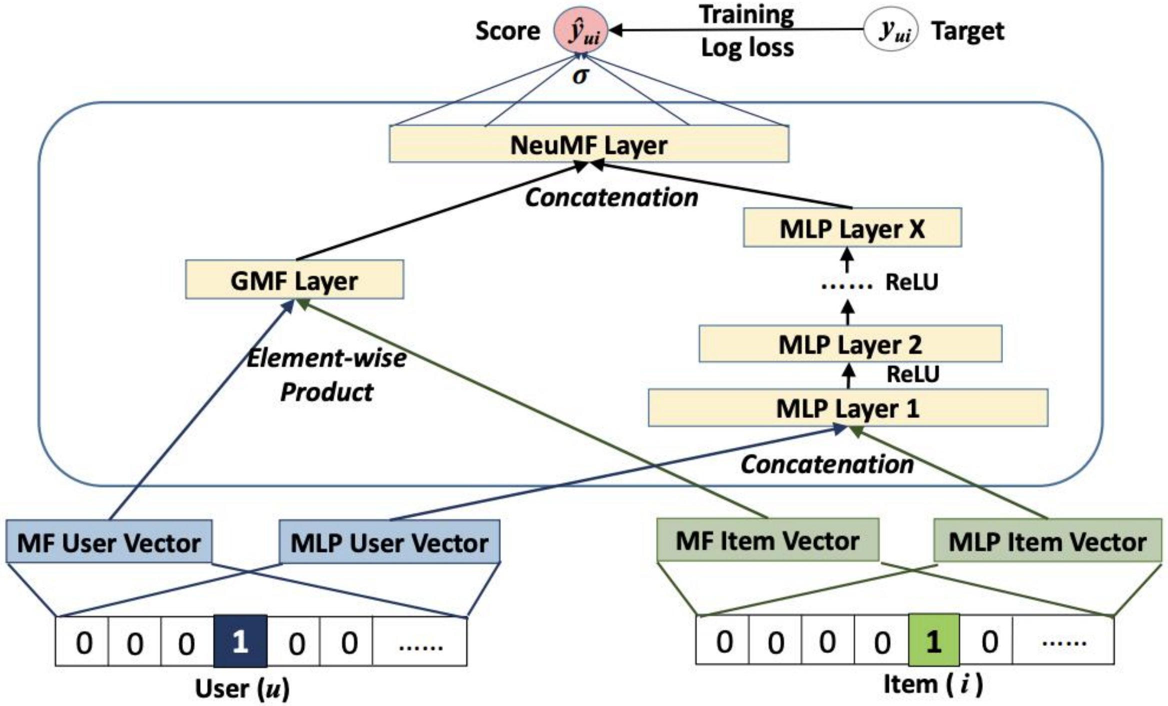
DeepCF
A Unified Framework of Representation Learning and Matching Function Learning in Recommender System

- Recommendation (match problem) : match proper items for proper users
    - representation learning-based CF methods. eg. MF/SVD/SVD++/DMF(IJCAI 2017，点积操作表达有限制)
    - matching function learning-based CF methods
    - representation and matching function learning-based CF methods. eg. DeepCF(AAAI 2019)/NCF(WWW 2017, Deep+Shallow模式)

DeepCF VS. NCF





- DeepCF放弃了传统的Deep + Shallow模式，而仅采用Deep模型来实现具有隐式反馈的协同过滤（NCF将ID作为输入，而DeepCF将交互矩阵作为输入）。

- 结合基于表示和匹配函数的学习
- 利用隐式反馈

GMF： CFNet-rl

- MF的扩展，用一层网络结构学习user以及item的线性关系。
- MLP作为表示函数
- 将交互矩阵作为输入
- 用element-wise product和参数化的网络层，来替代点积或余弦相似度

MLP： CFNet-ml

- 学习user以及item的非线性关系部分
- MLP来学习特征匹配函数
- 将交互矩阵作为输入

CFNet

- 合并CFNet-rl和CFNet-ml，得到最终的模型CFNet

样本构造+训练

- 从未观察到的交互中采样一些作为负实例
- 目标函数point-wise，交叉熵
- 预训练：可以分别预训练CFNet-rl和CFNet-ml，然后用它们来初始化CFNet
- mini-batch Adam、batch size 256、learning rate 0.001

评估

- NDCG：排序结果的评价指标。CG、DCG（考虑位置）、NDCG（标准化后的DCG）
- Hit Ratio(HR)： top-K推荐中衡量召回率的指标。
- 其他指标：
  - MAP

分析 - 结果分析

Table 2: Comparison results of different methods in terms of NDCG@10 and HR@10.

Datasets	Measures	Existing methods				CFNet			Improvement of
		ItemPop	eALS	DMF	NeuMF	CFNet-rl	CFNet-ml	CFNet	CFNet vs. NeuMF
ml-1m	HR	0.4535	0.7018	0.6565	**0.7210**	0.7127	0.7073	**0.7253**	0.6%
	NDCG	0.2542	0.4280	0.3761	**0.4387**	0.4336	0.4264	**0.4416**	0.7%
lastfm	HR	0.6628	0.8265	0.8840	**0.8868**	0.8840	0.8834	**0.8995**	1.4%
	NDCG	0.3862	0.5162	0.5804	**0.6007**	0.6001	0.5919	**0.6186**	3.0%
AMusic	HR	0.2483	0.3711	0.3744	0.3891	0.3947	**0.4071**	**0.4116**	5.8%
	NDCG	0.1304	0.2352	0.2149	0.2391	**0.2504**	0.2420	**0.2601**	8.8%
AToy	HR	0.2840	0.3717	0.3535	0.3650	0.3746	**0.3931**	**0.4150**	13.7%
	NDCG	0.1518	**0.2434**	0.2016	0.2155	0.2271	0.2293	**0.2513**	16.6%

- ItemPop：非个性化，根据热度排序
- eALS： SOTA MF
- DMF：SOTA 基于表示学习的MF方法
- NeuMF： SOTA 基于表示和匹配函数学习结合的MF方法

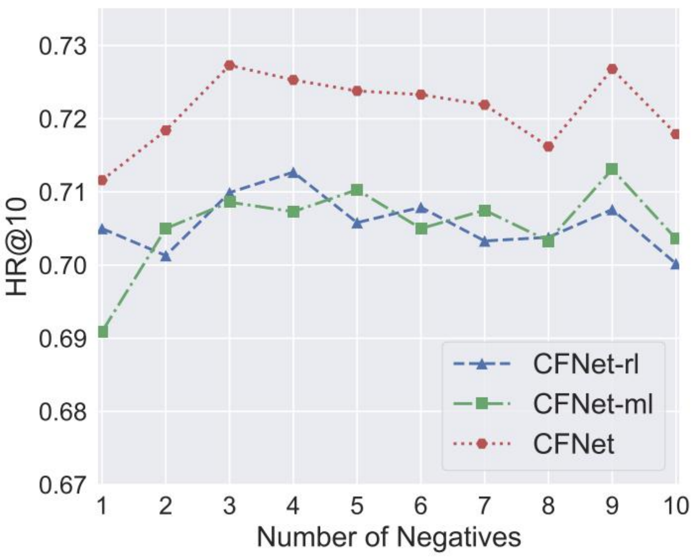
分析 – 预训练/超参数的影响

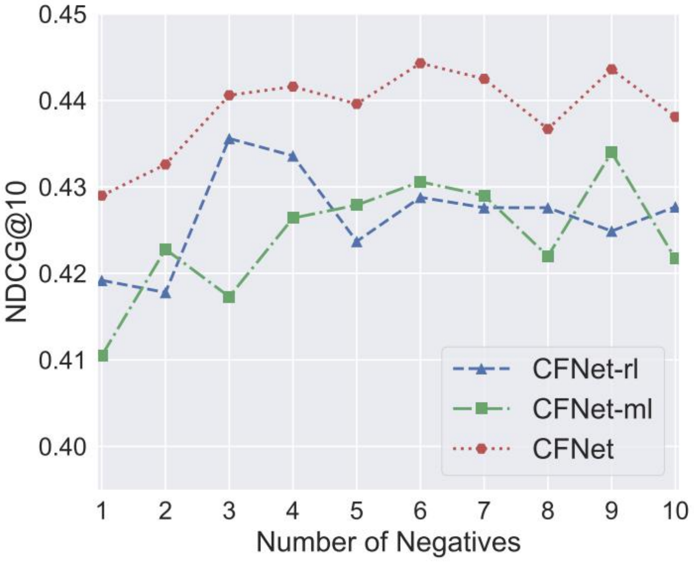
Table 3: Performance of CFNet with/without pre-training.

Datasets	Without pre-training HR	Without pre-training NDCG	With pre-training HR	With pre-training NDCG
ml-1m	0.6962	0.4222	0.7253	0.4416
lastfm	0.8685	0.5920	0.8995	0.6186
AMusic	0.3530	0.2204	0.4116	0.2601
AToy	0.3067	0.1653	0.4150	0.2513

Table 4: Performance of CFNet with different number of predictive factors.

Datasets	Measures	Dimensions of predictive vectors 8	16	32	64
ml-1m	HR	0.6820	0.6982	0.7157	0.7253
	NDCG	0.3992	0.4161	0.4351	0.4416
lastfm	HR	0.8840	0.8857	0.8937	0.8995
	NDCG	0.6049	0.6111	0.6143	0.6186
AMusic	HR	0.4003	0.4313	0.4262	0.4116
	NDCG	0.2480	0.2617	0.2661	0.2601
AToy	HR	0.3797	0.3902	0.4026	0.4150
	NDCG	0.2273	0.2331	0.2383	0.2513





未来工作

- auxiliary data、richer information
- 更好的聚合函数替换elementwise product、concatenation
- pairwise loss替换point-wise loss

代码

```
def get_model(train, num_users, num_items, userlayers, itemlayers, layers):
    dmf_num_layer = len(userlayers) #Number of layers in the DMF
    mlp_num_layer = len(layers) #Number of layers in the MLP
    user_matrix = K.constant(getTrainMatrix(train))
    item_matrix = K.constant(getTrainMatrix(train).T)
    # Input variables
    user_input = Input(shape=(1,), dtype='int32', name='user_input')
    item_input = Input(shape=(1,), dtype='int32', name='item_input')

    # Embedding layer
    user_rating= Lambda(lambda x: tf.gather(user_matrix, tf.to_int32(x)))(user_input)
    item_rating = Lambda(lambda x: tf.gather(item_matrix, tf.to_int32(x)))(item_input)
    user_rating = Reshape((num_items, ))(user_rating)
    item_rating = Reshape((num_users, ))(item_rating)
```

```python
# DMF part
userlayer = Dense(userlayers[0],  activation="linear" , name='user_layer0')
itemlayer = Dense(itemlayers[0], activation="linear" , name='item_layer0')
dmf_user_latent = userlayer(user_rating)
dmf_item_latent = itemlayer(item_rating)
for idx in range(1, dmf_num_layer):
    userlayer = Dense(userlayers[idx],  activation='relu', name='user_layer%d' % idx)
    itemlayer = Dense(itemlayers[idx],  activation='relu', name='item_layer%d' % idx)
    dmf_user_latent = userlayer(dmf_user_latent)
    dmf_item_latent = itemlayer(dmf_item_latent)
dmf_vector = multiply([dmf_user_latent, dmf_item_latent])

# MLP part
MLP_Embedding_User = Dense(layers[0]//2, activation="linear" , name='user_embedding')
MLP_Embedding_Item  = Dense(layers[0]//2, activation="linear" , name='item_embedding')
mlp_user_latent = MLP_Embedding_User(user_rating)
mlp_item_latent = MLP_Embedding_Item(item_rating)
mlp_vector = concatenate([mlp_user_latent, mlp_item_latent])
for idx in range(1, mlp_num_layer):
    layer = Dense(layers[idx], activation='relu', name="layer%d" % idx)
    mlp_vector = layer(mlp_vector)

# Concatenate DMF and MLP parts
predict_vector = concatenate([dmf_vector, mlp_vector])

# Final prediction layer
prediction = Dense(1, activation='sigmoid', kernel_initializer=initializers.lecun_normal(),
                   name="prediction")(predict_vector)

model_ = Model(inputs=[user_input, item_input],
               outputs=prediction)
```

- DMF: 点积操作表达有限制
- NCF: MLP来替换传统CF算法中的内积操作