In [30]:

```
import numpy as np
import tensorflow as tf
import theano
import theano.tensor as T
import keras
from keras import backend as K
from keras import initializers
from keras.regularizers import l1, l2
from keras.models import Sequential, Model
from keras.layers.core import Dense, Lambda, Activation
from keras.layers import Embedding, Input, Dense, merge, Reshape, Merge, Flatten, Dropout
from keras.optimizers import Adagrad, Adam, SGD, RMSprop
from evaluate import evaluate_model
from Dataset import Dataset
from time import time
import sys
import GMF, MLP
import argparse

import warnings
warnings.filterwarnings('ignore')
```

In [31]:

```
def get_train_instances(train, num_negatives):
    # Train -> sparse matrix (rows: 943, items: 1682; for ml-100k dataset)

    # Intialise the user_id, item_id and the labels (o or 1) lists.
    user_input, item_input, labels = [],[],[]

    # total number of users present in the dataset
    num_users = train.shape[0]

    # iterate over each non-zero rating.
    for (u, i) in train.keys():

        # append positive instance (non-zero rating)
        user_input.append(u)
        item_input.append(i)
        labels.append(1)

        # find and then append negative instances (zero rating)
        # total negative instance (user_id, item_id) = num_negatives
        for t in range(num_negatives):

            # generate a random item_id between [0, 1682] (Ex: for ml-100k dat
aset)
            j = np.random.randint(num_items)

            # check whether the generated item_id is present in the non-zero r
ating dataset or not.
            # If yes then we'll move inside the while loop and generate a new
 item_id and
            # then, check again till we generate a item_id which is not presen
t in the train dataset.
            while (u, j) in train:
                j = np.random.randint(num_items)

            # append the user_id, item_id (generated above.)
            user_input.append(u)
            item_input.append(j)

            # rating for the negative instance will be zero
            labels.append(0)

    return user_input, item_input, labels
```

In [32]:

```
def load_pretrain_model(model, gmf_model, mlp_model, num_layers):
    # MF embeddings
    gmf_user_embeddings = gmf_model.get_layer('user_embedding').get_weights()
    gmf_item_embeddings = gmf_model.get_layer('item_embedding').get_weights()
    model.get_layer('mf_embedding_user').set_weights(gmf_user_embeddings)
    model.get_layer('mf_embedding_item').set_weights(gmf_item_embeddings)

    # MLP embeddings
    mlp_user_embeddings = mlp_model.get_layer('user_embedding').get_weights()
    mlp_item_embeddings = mlp_model.get_layer('item_embedding').get_weights()
    model.get_layer('mlp_embedding_user').set_weights(mlp_user_embeddings)
    model.get_layer('mlp_embedding_item').set_weights(mlp_item_embeddings)

    # MLP layers
    for i in range(1, num_layers):
        mlp_layer_weights = mlp_model.get_layer('layer%d' %i).get_weights()
        model.get_layer('layer%d' %i).set_weights(mlp_layer_weights)

    # Prediction weights
    gmf_prediction = gmf_model.get_layer('prediction').get_weights()
    mlp_prediction = mlp_model.get_layer('prediction').get_weights()
    new_weights = np.concatenate((gmf_prediction[0], mlp_prediction[0]), axis=
0)
    new_b = gmf_prediction[1] + mlp_prediction[1]
    model.get_layer('prediction').set_weights([0.5*new_weights, 0.5*new_b])
    return model
```

Initialise the model arguments

In [41]:

```
num_epochs = 10
batch_size = 256
mf_dim = 8
layers = [64,32,16,8]
reg_mf = 0
reg_layers = [0,0,0,0]
num_negatives = 4
learning_rate = 0.001
learner = 'adam'
verbose = 1
mf_pretrain = ''
mlp_pretrain = ''
dataset = 'ml-100k'
path = 'Data/'
out = 0

topK = 10
evaluation_threads = 1#mp.cpu_count()
model_out_file = 'Pretrain/%s_NeuMF_%d_%s_%d.h5' %(dataset, mf_dim, layers, ti
me())
```

Loading data

In [42]:

```
t1 = time()
dataset = Dataset(path + dataset)
train, testRatings, testNegatives = dataset.trainMatrix, dataset.testRatings,
dataset.testNegatives
num_users, num_items = train.shape
print("Load data done [%.1f s]. #user=%d, #item=%d, #train=%d, #test=%d"
      %(time()-t1, num_users, num_items, train.nnz, len(testRatings)))
```

```
Load data done [1.2 s]. #user=944, #item=1682, #train=99057, #test=943
```

In [43]:

```
def get_model(num_users, num_items, mf_dim=10, layers=[10], reg_layers=[0], re
g_mf=0):
    assert len(layers) == len(reg_layers)
    num_layer = len(layers) #Number of layers in the MLP
    # Input variables
    shape=(1,)
    user_input = Input(shape, name = 'user_input')
    item_input = Input(shape, name = 'item_input')

    # Embedding layer
    MF_Embedding_User = Embedding(num_users,
                                  mf_dim,
                                  name = 'mf_embedding_user',
                                  embeddings_initializer = 'random_normal',
                                  W_regularizer = l2(reg_mf),
                                  input_length=1)

    MF_Embedding_Item = Embedding(num_items,
                                  mf_dim,
                                  name = 'mf_embedding_item',
                                  embeddings_initializer = 'random_normal',
                                  W_regularizer = l2(reg_mf),
                                  input_length=1)

    neurons = int(layers[0]/2)
    MLP_Embedding_User = Embedding(num_users,
                                   neurons,
                                   name = "mlp_embedding_user",
                                   embeddings_initializer = 'random_normal',
                                   W_regularizer = l2(reg_layers[0]),
                                   input_length=1)

    MLP_Embedding_Item = Embedding(num_items,
                                   neurons,
                                   name = 'mlp_embedding_item',
                                   embeddings_initializer = 'random_normal',
                                   W_regularizer = l2(reg_layers[0]),
                                   input_length=1)

    # MF part
    mf_user_latent = Flatten()(MF_Embedding_User(user_input))
    mf_item_latent = Flatten()(MF_Embedding_Item(item_input))
    mf_vector = merge([mf_user_latent, mf_item_latent], mode = 'mul') # elemen
t-wise multiply

    # MLP part
    mlp_user_latent = Flatten()(MLP_Embedding_User(user_input))
    mlp_item_latent = Flatten()(MLP_Embedding_Item(item_input))
    mlp_vector = merge([mlp_user_latent, mlp_item_latent], mode = 'concat')
    for idx in range(1, num_layer):
        layer = Dense(layers[idx], W_regularizer= l2(reg_layers[idx]), activat
ion='relu', name="layer%d" %idx)
        mlp_vector = layer(mlp_vector)

    # Concatenate MF and MLP parts
    #mf_vector = Lambda(lambda x: x * alpha)(mf_vector)
    #mlp_vector = Lambda(lambda x : x * (1-alpha))(mlp_vector)
    predict_vector = merge([mf_vector, mlp_vector], mode = 'concat')

    # Final prediction layer
    prediction = Dense(1, activation='sigmoid', init='lecun_uniform', name =
"prediction")(predict_vector)

    model = Model(input=[user_input, item_input],
                  output=prediction)

    return model
```

Build model

In [44]:

```
model = get_model(int(num_users), int(num_items), mf_dim, layers, reg_layers,
reg_mf)
if learner.lower() == "adagrad":
    model.compile(optimizer=Adagrad(lr=learning_rate), loss='binary_crossentro
py')
elif learner.lower() == "rmsprop":
    model.compile(optimizer=RMSprop(lr=learning_rate), loss='binary_crossentro
py')
elif learner.lower() == "adam":
    model.compile(optimizer=Adam(lr=learning_rate), loss='binary_crossentropy'
)
else:
    model.compile(optimizer=SGD(lr=learning_rate), loss='binary_crossentropy')
```

Init performance

In [45]:

```
(hits, ndcgs) = evaluate_model(model, testRatings, testNegatives, topK, evalua
tion_threads)
hr, ndcg = np.array(hits).mean(), np.array(ndcgs).mean()
print('Init: HR = %.4f, NDCG = %.4f' % (hr, ndcg))
best_hr, best_ndcg, best_iter = hr, ndcg, -1
if out > 0:
    model.save_weights(model_out_file, overwrite=True)
```

```
Init: HR = 0.1060, NDCG = 0.0446
```

Training model

In [46]:

```
for epoch in range(num_epochs):
    t1 = time()
    # Generate training instances
    user_input, item_input, labels = get_train_instances(train, num_negatives)

    # Training
    hist = model.fit([np.array(user_input), np.array(item_input)], #input
                     np.array(labels), # labels
                     batch_size=batch_size,
                     nb_epoch=1,
                     verbose=0,
                     shuffle=True)
    t2 = time()

    # Evaluation
    if epoch % verbose == 0:
        (hits, ndcgs) = evaluate_model(model, testRatings, testNegatives, topK
, evaluation_threads)
        hr, ndcg, loss = np.array(hits).mean(), np.array(ndcgs).mean(), hist.h
istory['loss'][0]
        print('Iteration %d [%.1f s]: HR = %.4f, NDCG = %.4f, loss = %.4f [%.1
f s]'
              % (epoch,  t2-t1, hr, ndcg, loss, time()-t2))
        if hr > best_hr:
            best_hr, best_ndcg, best_iter = hr, ndcg, epoch
            if out > 0:
                model.save_weights(model_out_file, overwrite=True)
```

```
Iteration 0 [8.9 s]: HR = 0.4624, NDCG = 0.2599, loss = 0.3711 [1.1 s]
Iteration 1 [6.4 s]: HR = 0.5885, NDCG = 0.3267, loss = 0.3015 [1.0 s]
Iteration 2 [6.0 s]: HR = 0.6013, NDCG = 0.3427, loss = 0.2743 [1.1 s]
Iteration 3 [6.8 s]: HR = 0.6320, NDCG = 0.3560, loss = 0.2596 [1.0 s]
Iteration 4 [5.8 s]: HR = 0.6416, NDCG = 0.3660, loss = 0.2507 [0.8 s]
Iteration 5 [5.6 s]: HR = 0.6522, NDCG = 0.3683, loss = 0.2420 [0.8 s]
Iteration 6 [5.6 s]: HR = 0.6554, NDCG = 0.3702, loss = 0.2355 [0.8 s]
Iteration 7 [5.5 s]: HR = 0.6649, NDCG = 0.3805, loss = 0.2303 [0.8 s]
Iteration 8 [5.6 s]: HR = 0.6596, NDCG = 0.3828, loss = 0.2244 [0.8 s]
Iteration 9 [5.5 s]: HR = 0.6755, NDCG = 0.3867, loss = 0.2188 [0.8 s]
```

In [47]:

```
print("End. Best Iteration %d:  HR = %.4f, NDCG = %.4f. " %(best_iter, best_hr
, best_ndcg))
if out > 0:
    print("The best NeuMF model is saved to %s" %(model_out_file))
```

```
End. Best Iteration 9:  HR = 0.6755, NDCG = 0.3867.
```

Recommendations

After training of the model is done.

In [60]:

```
RATINGS_CSV_FILE = r'C:\Main Project\ml-100k\u.data'
USERS_CSV_FILE = r'C:\Main Project\ml-100k\u.user'
MOVIES_CSV_FILE = r'C:\Main Project\ml-100k\u.item'
K_FACTORS = mf_dim
TEST_USER = 944
```

Load Ratings Dataset

In [61]:

```
import pandas as pd

ratings = pd.read_csv(RATINGS_CSV_FILE, sep='\t', names=['userid', 'movieid',
'rating', 'timestamp'])
max_userid = ratings['userid'].drop_duplicates().max()
max_movieid = ratings['movieid'].drop_duplicates().max()
ratings['user_emb_id'] = ratings['userid'] - 1
ratings['movie_emb_id'] = ratings['movieid'] - 1
print(len(ratings), 'ratings loaded.')
ratings.head()
```

```
100000 ratings loaded.
```

Out[61]:

	userid	movieid	rating	timestamp	user_emb_id	movie_emb_id
0	196	242	3	881250949	195	241
1	186	302	3	891717742	185	301
2	22	377	1	878887116	21	376
3	244	51	2	880606923	243	50
4	166	346	1	886397596	165	345

Load Users Dataset

In [62]:

```
users = pd.read_csv(USERS_CSV_FILE, sep='|', names=['userid', 'age_desc', 'gender', 'occ_desc', 'zipcode'])
print(len(users), 'descriptions of', max_userid, 'users loaded.')
users.head()
```

```
943 descriptions of 943 users loaded.
```

Out[62]:

	userid	age_desc	gender	occ_desc	zipcode
0	1	24	M	technician	85711
1	2	53	F	other	94043
2	3	23	M	writer	32067
3	4	24	M	technician	43537
4	5	33	F	other	15213

Load Movies Dataset

```
In [63]: movies = pd.read_csv(MOVIES_CSV_FILE,
                              sep='|',
                              encoding='latin-1',
                              names=["movieid", "title", "release_date", "video_release
         _date",
                                     "IMDb_URL", "unknown", "Action", "Adventure", "Ani
         mation",
                                     "Children's", "Comedy", "Crime", "Documentary", "D
         rama", "Fantasy",
                                     "Film-Noir", "Horror", "Musical", "Mystery", "Roma
         nce", "Sci-Fi",
                                     "Thriller", "War", "Western", "genre"])
         print(len(movies), 'descriptions of', max_movieid, 'movies loaded.')
         movies.head()
```

```
1682 descriptions of 1682 movies loaded.
```

Out[63]:

	movieid	title	release_date	video_release_date	IMDb_URL	unkno
0	1	Toy Story (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Toy%20Story%2...	0
1	2	GoldenEye (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?GoldenEye%20(...	0
2	3	Four Rooms (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Four%20Rooms%...	0
3	4	Get Shorty (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Get%20Shorty%...	0
4	5	Copycat (1995)	01-Jan-1995	NaN	http://us.imdb.com/M/title-exact?Copycat%20(1995)	0

5 rows × 25 columns

Generate Genres column

In [64]:
```
genres_list = {'unknown':0, 'Action':1, 'Adventure':2, 'Animation':3, "Childre
n's":4, 'Comedy':5, 'Crime':6, 'Documentary':7,
          'Drama':8, 'Fantasy':9, 'Film-Noir':10, 'Horror':11, 'Musical':12,
'Mystery':13, 'Romance':14, 'Sci-Fi':15,
          'Thriller':16, 'War':17, 'Western':18}
for i, row in movies.iterrows():
    temp_genre = ''
    for key, value in genres_list.items():
        if row.loc[key] == 1:
            temp_genre += key + '|'
    movies.loc[i, 'genre'] = temp_genre
movies = movies.drop(["release_date", "video_release_date", "IMDb_URL", "unkno
wn", "Action",
                      "Adventure", "Animation", "Children's", "Comedy", "Crim
e", "Documentary",
                      "Drama", "Fantasy", "Film-Noir", "Horror", "Musical", "M
ystery", "Romance",
                      "Sci-Fi", "Thriller", "War", "Western"], axis = 1)
movies.head()
```

Out[64]:

	movieid	title	genre
0	1	Toy Story (1995)	Animation|Children's|Comedy|
1	2	GoldenEye (1995)	Action|Adventure|Thriller|
2	3	Four Rooms (1995)	Thriller|
3	4	Get Shorty (1995)	Action|Comedy|Drama|
4	5	Copycat (1995)	Crime|Drama|Thriller|

model_out_file = 'Pretrain/ml-100k_NeuMF_8_[64, 32, 16, 8]_1521013085.h5' model.load_weights(model_out_file)

In [65]:
```
users[users['userid'] == TEST_USER]
```

Out[65]:

	userid	age_desc	gender	occ_desc	zipcode

In [66]:
```
def predict_rating(userid, movieid):
    val = model.predict([np.array([userid - 1]), np.array([movieid - 1])])[0][
0]
    sys.stdout.write('\r[{}, {}] : prediction = {}'.format(
        userid,
        movieid,
        val
    ))
    sys.stdout.flush()
    return val
```

In [67]:
```
# get all the (TEST_USER, item) ratings (non-zero ones specifically)
user_ratings = ratings[ratings['userid'] == TEST_USER][['userid', 'movieid',
'rating']]
```

In [68]:

```
recommendations = ratings[ratings['movieid'].isin(user_ratings['movieid']) ==
False][['movieid']].drop_duplicates()
recommendations['prediction'] = recommendations.apply(lambda x: predict_rating
(TEST_USER, x['movieid']), axis=1)
recommendations.sort_values(by='prediction',
                          ascending=False).merge(movies,
                                                   on='movieid',
                                                   how='inner',
                                                   suffixes=['_u', '_m']).head(1
0)
```

```
[944, 1641] : prediction = 1.7591331925359555e-05
```

Out[68]:

	movieid	prediction	title	genre
0	100	0.946090	Fargo (1996)	Crime|Drama|Thriller|
1	56	0.921848	Pulp Fiction (1994)	Crime|Drama|
2	50	0.911081	Star Wars (1977)	Action|Adventure|Romance|Sci-Fi|War|
3	181	0.875971	Return of the Jedi (1983)	Action|Adventure|Romance|Sci-Fi|War|
4	7	0.857967	Twelve Monkeys (1995)	Drama|Sci-Fi|
5	288	0.855101	Scream (1996)	Horror|Thriller|
6	121	0.848511	Independence Day (ID4) (1996)	Action|Sci-Fi|War|
7	174	0.843846	Raiders of the Lost Ark (1981)	Action|Adventure|
8	98	0.820350	Silence of the Lambs, The (1991)	Drama|Thriller|
9	286	0.820196	English Patient, The (1996)	Drama|Romance|War|