个性化推荐

本教程源代码目录在book/recommender_system,初次使用请您参考Book文档使用说明。

说明:

1. 硬件环境要求： 本文可支持在CPU、GPU下运行
2. Docker镜像支持的CUDA/cuDNN版本： 如果使用了Docker运行Book，请注意：这里所提供的默认镜像的GPU环境为 CUDA 8/cuDNN 5，对于NVIDIA Tesla V100等要求CUDA 9的 GPU，使用该镜像可能会运行失败。
3. 文档和脚本中代码的一致性问题： 请注意：为使本文更加易读易用，我们拆分、调整了train.py的代码并放入本文。本文中代码与train.py的运行结果一致，可直接运行train.py进行验证。

背景介绍

在网络技术不断发展和电子商务规模不断扩大的背景下，商品数量和种类快速增长，用户需要花费大量时间才能找到自己想买的商品，这就是信息超载问题。为了解决这个难题，个性化推荐系统（Recommender System）应运而生。

个性化推荐系统是信息过滤系统（Information Filtering System）的子集，它可以用在很多领域，如电影、音乐、电商和 Feed 流推荐等。个性化推荐系统通过分析、挖掘用户行为，发现用户的个性化需求与兴趣特点，将用户可能感兴趣的信息或商品推荐给用户。与搜索引擎不同，个性化推荐系统不需要用户准确地描述出自己的需求，而是根据用户的历史行为进行建模，主动提供满足用户兴趣和需求的信息。

1994年明尼苏达大学推出的GroupLens系统[1]一般被认为是个性化推荐系统成为一个相对独立的研究方向的标志。该系统首次提出了基于协同过滤来完成推荐任务的思想，此后，基于该模型的协同过滤推荐引领了个性化推荐系统十几年的发展方向。

传统的个性化推荐系统方法主要有：

- 协同过滤推荐（Collaborative Filtering Recommendation）：该方法是应用最广泛的技术之一，需要收集和分析用户的历史行为、活动和偏好。它通常可以分为两个子类：基于用户（User-Based）的推荐[1] 和基于物品（Item-Based）的推荐[2]。该方法的一个关键优势是它不依赖于机器去分析物品的内容特征，因此它无需理解物品本身也能够准确地推荐诸如电影之类的复杂物品；缺点是对于没有任何行为的新用户存在冷启动的问题，同时也存在用户与商品之间的交互数据不够多造成的稀疏问题。值得一提的是，社交网络[3]或地理位置等上下文信息都可以结合到协同过滤中去。
- 基于内容过滤推荐[4]（Content-based Filtering Recommendation）：该方法利用商品的内容描述，抽象出有意义的特征，通过计算用户的兴趣和商品描述之间的相似度，来给用户做推荐。优点是简单直接，不需要依据其他用户对商品的评价，而是通过商品属性进行商品相似度度量，从而推荐给用户所感兴趣商品的相似商品；缺点是对于没有任何行为的新用户同样存在冷启动的问题。
- 组合推荐[5]（Hybrid Recommendation）：运用不同的输入和技术共同进行推荐，以弥补各自推荐技术的缺点。

近些年来，深度学习在很多领域都取得了巨大的成功。学术界和工业界都在尝试将深度学习应用于个性化推荐系统领域中。深度学习具有优秀的自动提取特征的能力，能够学习多层次的抽象特征表示，并对异质或跨域的内容信息进行学习，可以一定程度上处理个性化推荐系统冷启动问题[6]。本教程主要介绍个性化推荐的深度学习模型，以及如何使用PaddlePaddle实现模型。

效果展示

我们使用包含用户信息、电影信息与电影评分的数据集作为个性化推荐的应用场景。当我们训练好模型后，只需要输入对应的用户ID和电影ID，就可以得出一个匹配的分数（范围[0,5]，分数越高视为兴趣越大），然后根据所有电影的推荐得分排序，推荐给用户可能感兴趣的电影。

```
Input movie_id: 1962
Input user_id: 1
Prediction Score is 4.25
```

本章中，我们首先介绍YouTube的视频个性化推荐系统[7]，然后介绍我们实现的融合推荐模型。

YouTube的深度神经网络个性化推荐系统

YouTube是世界上最大的视频上传、分享和发现网站，YouTube个性化推荐系统为超过10亿用户从不断增长的视频库中推荐个性化的内容。整个系统由两个神经网络组成：候选生成网络和排序网络。候选生成网络从百万量级的视频库中生成上百个候选，排序网络对候选进行打分排序，输出排名最高的数十个结果。系统结构如图1所示：

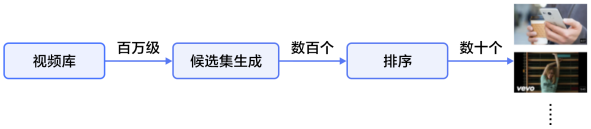

图1. YouTube 个性化推荐系统结构

候选生成网络（Candidate Generation Network）

候选生成网络将推荐问题建模为一个类别数极大的多类分类问题：对于一个Youtube用户，使用其观看历史（视频ID）、搜索词记录（search tokens）、人口学信息（如地理位置、用户登录设备）、二值特征（如性别，是否登录）和连续特征（如用户年龄）等，对视频库中所有视频进行多分类，得到每一类别的分类结果（即每一个视频的推荐概率），最终输出概率较高的几百个视频。

首先，将观看历史及搜索词记录这类历史信息，映射为向量后取平均值得到定长表示；同时，输入人口学特征以优化新用户的推荐效果，并将二值特征和连续特征归一化处理到[0, 1]范围。接下来，将所有特征表示拼接为一个向量，并输入给非线形多层感知器（MLP，详见识别数字教程）处理。最后，训练时将MLP的输出给softmax做分类，预测时计算用户的综合特征（MLP的输出）与所有视频的相似度，取得分最高的k个作为候选生成网络的筛选结果。图2显示了候选生成网络结构。

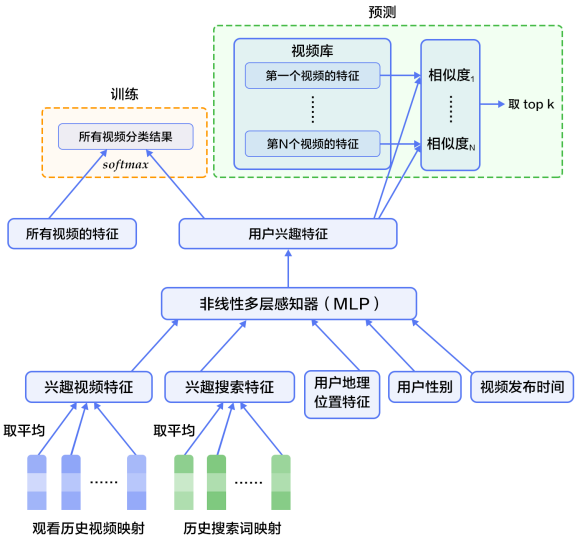

图2. 候选生成网络结构

对于一个用户U，预测此刻用户要观看的视频ω为视频i的概率公式为：

$$P(\omega = i|u) = \frac{e^{v_i u}}{\sum_{j \in V} e^{v_j u}}$$

其中u为用户U的特征表示，V为视频库集合，v_i为视频库中第i个视频的特征表示。u和v_i为长度相等的向量，两者点积可以通过全连接层实现。

考虑到softmax分类的类别数非常多，为了保证一定的计算效率：1）训练阶段，使用负样本类别采样将实际计算的类别数缩小至数千；2）推荐（预测）阶段，忽略softmax的归一化计算（不影响结果），将类别打分问题简化为点积（dot product）空间中的最近邻（nearest neighbor）搜索问题，取与u最近的k个视频作为生成的候选。

排序网络（Ranking Network）

排序网络的结构类似于候选生成网络，但是它的目标是对候选进行更细致的打分排序。和传统广告排序中的特征抽取方法类似，这里也构造了大量的用于视频排序的相关特征（如视频 ID、上次观看时间等）。这些特

融合推荐模型

本节会使用卷积神经网络（Convolutional Neural Networks）来学习电影名称的表示。下面会依次介绍文本卷积神经网络以及融合推荐模型。

文本卷积神经网络（CNN）

卷积神经网络经常用来处理具有类似网格拓扑结构（grid-like topology）的数据。例如，图像可以视为二维网格的像素点，自然语言可以视为一维的词序列。卷积神经网络可以提取多种局部特征，并对其进行组合抽象得到更高级的特征表示。实验表明，卷积神经网络能高效地对图像及文本问题进行建模处理。

卷积神经网络主要由卷积（convolution）和池化（pooling）操作构成，其应用及组合方式灵活多变，种类繁多。本小结我们以如图3所示的网络进行讲解：

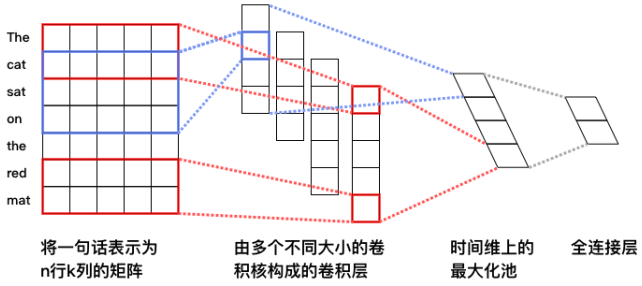

图3. 卷积神经网络文本分类模型

假设待处理句子的长度为n，其中第i个词的词向量为$x_i \in \mathbb{R}^k$，k为维度大小。

首先，进行词向量的拼接操作：将每h个词拼接起来形成一个大小为h的词窗口，记为$x_{i:i+h-1}$，它表示词序列$x_i, x_{i+1}, \ldots, x_{i+h-1}$的拼接，其中，$i$表示词窗口中第一个词在整个句子中的位置，取值范围从1到$n-h+1$，$x_{i:i+h-1} \in \mathbb{R}^{hk}$。

其次，进行卷积操作：把卷积核(kernel)$w \in \mathbb{R}^{hk}$应用于包含h个词的窗口$x_{i:i+h-1}$，得到特征$c_i = f(w \cdot x_{i:i+h-1} + b)$，其中$b \in \mathbb{R}$为偏置项（bias），$f$为非线性激活函数，如$sigmoid$。将卷积核应用于句子中所有的词窗口$x_{1:h}, x_{2:h+1}, \ldots, x_{n-h+1:n}$，产生一个特征图（feature map）：

$$c = [c_1, c_2, \ldots, c_{n-h+1}], c \in \mathbb{R}^{n-h+1}$$

接下来，对特征图采用时间维度上的最大池化（max pooling over time）操作得到此卷积核对应的整句话的特征$\hat{c}$，它是特征图中所有元素的最大值：

$$\hat{c} = max(c)$$

融合推荐模型概览

在融合推荐模型的电影个性化推荐系统中：

1. 首先，使用用户特征和电影特征作为神经网络的输入，其中：
   - 用户特征融合了四个属性信息，分别是用户ID、性别、职业和年龄。
   - 电影特征融合了三个属性信息，分别是电影ID、电影类型ID和电影名称。
2. 对用户特征，将用户ID映射为维度大小为256的向量表示，输入全连接层，并对其他三个属性也做类似的处理。然后将四个属性的特征表示分别全连接并相加。
3. 对电影特征，将电影ID以类似用户ID的方式进行处理，电影类型ID以向量的形式直接输入全连接层，电影名称用文本卷积神经网络得到其定长向量表示。然后将三个属性的特征表示分别全连接并相加。
4. 得到用户和电影的向量表示后，计算二者的余弦相似度作为个性化推荐系统的打分。最后，用该相似度打分和用户真实打分的差异的平方作为该回归模型的损失函数。

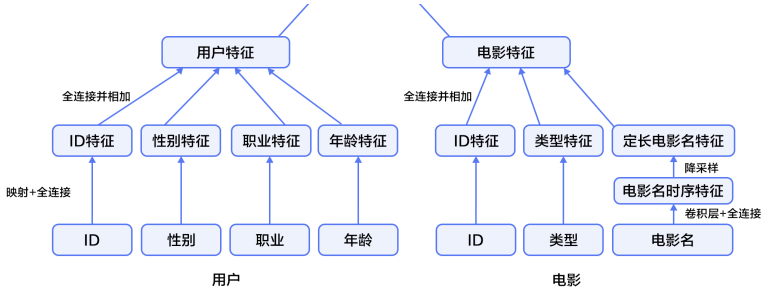

图4. 融合推荐模型

数据准备

数据介绍与下载

我们以 MovieLens 百万数据集（ml-1m）为例进行介绍。ml-1m 数据集包含了 6,000 位用户对 4,000 部电影的 1,000,000 条评价（评分范围 1~5 分，均为整数），由 GroupLens Research 实验室搜集整理。

Paddle在API中提供了自动加载数据的模块。数据模块为 `paddle.dataset.movielens`

```
import paddle
movie_info = paddle.dataset.movielens.movie_info()
print movie_info.values()[0]
```

```
# Run this block to show dataset's documentation
# help(paddle.dataset.movielens)
```

在原始数据中包含电影的特征数据，用户的特征数据，和用户对电影的评分。

例如，其中某一个电影特征为:

```
movie_info = paddle.dataset.movielens.movie_info()
print movie_info.values()[0]
```

```
<MovieInfo id(1), title(Toy Story ), categories(['Animation', "Children's", 'Comedy'])>
```

这表示，电影的id是1，标题是《Toy Story》，该电影被分为到三个类别中。这三个类别是动画，儿童，喜剧。

```
user_info = paddle.dataset.movielens.user_info()
print user_info.values()[0]
```

```
<UserInfo id(1), gender(F), age(1), job(10)>
```

这表示，该用户ID是1，女性，年龄比18岁还年轻。职业ID是10。

其中，年龄使用下列分布

- 1: "Under 18"
- 18: "18-24"
- 25: "25-34"
- 35: "35-44"
- 45: "45-49"
- 50: "50-55"
- 56: "56+"

职业是从下面几种选项里面选则得出:

- 0: "other" or not specified

- 3: "clerical/admin"
- 4: "college/grad student"
- 5: "customer service"
- 6: "doctor/health care"
- 7: "executive/managerial"
- 8: "farmer"
- 9: "homemaker"
- 10: "K-12 student"
- 11: "lawyer"
- 12: "programmer"
- 13: "retired"
- 14: "sales/marketing"
- 15: "scientist"
- 16: "self-employed"
- 17: "technician/engineer"
- 18: "tradesman/craftsman"
- 19: "unemployed"
- 20: "writer"

而对于每一条训练/测试数据，均为 <用户特征> + <电影特征> + 评分。

例如，我们获得第一条训练数据:

```
train_set_creator = paddle.dataset.movielens.train()
train_sample = next(train_set_creator())
uid = train_sample[0]
mov_id = train_sample[len(user_info[uid].value())]
print "User %s rates Movie %s with Score %s"%(user_info[uid], movie_info[mov_id], train_sample[-
```

```
User <UserInfo id(1), gender(F), age(1), job(10)> rates Movie <MovieInfo id(1193), title(One Fle
```

即用户1对电影1193的评价为5分。

模型配置说明

下面我们开始根据输入数据的形式配置模型。首先引入所需的库函数以及定义全局变量。

- IS_SPARSE: embedding中是否使用稀疏更新
- PASS_NUM: epoch数量

```
from __future__ import print_function
import math
import sys
import numpy as np
import paddle
import paddle.fluid as fluid
import paddle.fluid.layers as layers
import paddle.fluid.nets as nets

IS_SPARSE = True
BATCH_SIZE = 256
PASS_NUM = 20
```

然后为我们的用户特征综合模型定义模型配置

```
def get_usr_combined_features():
    """network definition for user part"""

    USR_DICT_SIZE = paddle.dataset.movielens.max_user_id() + 1

    uid = layers.data(name='user_id', shape=[1], dtype='int64')

    usr_emb = layers.embedding(
        input=uid,
        dtype='float32',
        size=[USR_DICT_SIZE, 32],
```

```
    usr_fc = layers.fc(input=usr_combined, size=32)

    USR_GENDER_DICT_SIZE = 2

    usr_gender_id = layers.data(name='gender_id', shape=[1], dtype='int64')

    usr_gender_emb = layers.embedding(
        input=usr_gender_id,
        size=[USR_GENDER_DICT_SIZE, 16],
        param_attr='gender_table',
        is_sparse=IS_SPARSE)

    usr_gender_fc = layers.fc(input=usr_gender_emb, size=16)

    USR_AGE_DICT_SIZE = len(paddle.dataset.movielens.age_table)
    usr_age_id = layers.data(name='age_id', shape=[1], dtype="int64")

    usr_age_emb = layers.embedding(
        input=usr_age_id,
        size=[USR_AGE_DICT_SIZE, 16],
        is_sparse=IS_SPARSE,
        param_attr='age_table')

    usr_age_fc = layers.fc(input=usr_age_emb, size=16)

    USR_JOB_DICT_SIZE = paddle.dataset.movielens.max_job_id() + 1
    usr_job_id = layers.data(name='job_id', shape=[1], dtype="int64")

    usr_job_emb = layers.embedding(
        input=usr_job_id,
        size=[USR_JOB_DICT_SIZE, 16],
        param_attr='job_table',
        is_sparse=IS_SPARSE)

    usr_job_fc = layers.fc(input=usr_job_emb, size=16)

    concat_embed = layers.concat(
        input=[usr_fc, usr_gender_fc, usr_age_fc, usr_job_fc], axis=1)

    usr_combined_features = layers.fc(input=concat_embed, size=200, act="tanh")

    return usr_combined_features
```

如上述代码所示，对于每个用户，我们输入4维特征。其中包括user_id,gender_id,age_id,job_id。这几维特征均是简单的整数值。为了后续神经网络处理这些特征方便，我们借鉴NLP中的语言模型，将这几维离散的整数值，变换成embedding取出。分别形成usr_emb, usr_gender_emb, usr_age_emb, usr_job_emb。

然后，我们对于所有的用户特征，均输入到一个全连接层(fc)中。将所有特征融合为一个200维度的特征。

进而，我们对每一个电影特征做类似的变换，网络配置为:

```
def get_mov_combined_features():
    """network definition for item(movie) part"""

    MOV_DICT_SIZE = paddle.dataset.movielens.max_movie_id() + 1

    mov_id = layers.data(name='movie_id', shape=[1], dtype='int64')

    mov_emb = layers.embedding(
        input=mov_id,
        dtype='float32',
        size=[MOV_DICT_SIZE, 32],
        param_attr='movie_table',
        is_sparse=IS_SPARSE)

    mov_fc = layers.fc(input=mov_emb, size=32)

    CATEGORY_DICT_SIZE = len(paddle.dataset.movielens.movie_categories())

    category_id = layers.data(
        name='category_id', shape=[1], dtype='int64', lod_level=1)

    mov_categories_emb = layers.embedding(
        input=category_id, size=[CATEGORY_DICT_SIZE, 32], is_sparse=IS_SPARSE)

    mov_categories_hidden = layers.sequence_pool(
        input=mov_categories_emb, pool_type="sum")

    MOV_TITLE_DICT_SIZE = len(paddle.dataset.movielens.get_movie_title_dict())

    mov_title_id = layers.data(
        name='movie_title', shape=[1], dtype='int64', lod_level=1)

    mov_title_emb = layers.embedding(
        input=mov_title_id, size=[MOV_TITLE_DICT_SIZE, 32], is_sparse=IS_SPARSE)
```

```
        num_filters=32,
        filter_size=3,
        act="tanh",
        pool_type="sum")

    concat_embed = layers.concat(
        input=[mov_fc, mov_categories_hidden, mov_title_conv], axis=1)

    mov_combined_features = layers.fc(input=concat_embed, size=200, act="tanh")

    return mov_combined_features
```

电影标题名称(title)是一个序列的整数，整数代表的是这个词在索引序列中的下标。这个序列会被送入 `sequence_conv_pool` 层，这个层会在时间维度上使用卷积和池化。因为如此，所以输出会是固定长度，尽管输入的序列长度各不相同。

最后，我们定义一个 `inference_program` 来使用余弦相似度计算用户特征与电影特征的相似性。

```
def inference_program():
    """the combined network"""

    usr_combined_features = get_usr_combined_features()
    mov_combined_features = get_mov_combined_features()

    inference = layers.cos_sim(X=usr_combined_features, Y=mov_combined_features)
    scale_infer = layers.scale(x=inference, scale=5.0)

    return scale_infer
```

进而，我们定义一个 `train_program` 来使用 `inference_program` 计算出的结果，在标记数据的帮助下来计算误差。我们还定义了一个 `optimizer_func` 来定义优化器。

```
def train_program():
    """define the cost function"""

    scale_infer = inference_program()

    label = layers.data(name='score', shape=[1], dtype='float32')
    square_cost = layers.square_error_cost(input=scale_infer, label=label)
    avg_cost = layers.mean(square_cost)

    return [avg_cost, scale_infer]

def optimizer_func():
    return fluid.optimizer.SGD(learning_rate=0.2)
```

训练模型

定义训练环境

定义您的训练环境，可以指定训练是发生在CPU还是GPU上。

```
use_cuda = False
place = fluid.CUDAPlace(0) if use_cuda else fluid.CPUPlace()
```

定义数据提供器

下一步是为训练和测试定义数据提供器。提供器读入一个大小为 `BATCH_SIZE` 的数据。 `paddle.dataset.movielens.train` 每次会在乱序化后提供一个大小为 `BATCH_SIZE` 的数据，乱序化的大小为缓存大小 `buf_size` 。

```
train_reader = paddle.batch(
    paddle.reader.shuffle(
        paddle.dataset.movielens.train(), buf_size=8192),
    batch_size=BATCH_SIZE)

test_reader = paddle.batch(
    paddle.dataset.movielens.test(), batch_size=BATCH_SIZE)
```

提供数据

`feed_order` 用来定义每条产生的数据和 `paddle.layer.data` 之间的映射关系。比如， `movielens.train` 产生的第一列的数据对应的是 `user_id` 这个特征。

```
feed_order = [
    'user_id', 'gender_id', 'age_id', 'job_id', 'movie_id', 'category_id',
    'movie_title', 'score'
]
```

构建训练程序以及测试程序

分别构建训练程序和测试程序，并引入训练优化器。

```
main_program = fluid.default_main_program()
star_program = fluid.default_startup_program()
[avg_cost, scale_infer] = train_program()

test_program = main_program.clone(for_test=True)
sgd_optimizer = optimizer_func()
sgd_optimizer.minimize(avg_cost)
exe = fluid.Executor(place)

def train_test(program, reader):
    count = 0
    feed_var_list = [
        program.global_block().var(var_name) for var_name in feed_order
    ]
    feeder_test = fluid.DataFeeder(
    feed_list=feed_var_list, place=place)
    test_exe = fluid.Executor(place)
    accumulated = 0
    for test_data in reader():
        avg_cost_np = test_exe.run(program=program,
                                               feed=feeder_test.feed(test_data),
                                               fetch_list=[avg_cost])
        accumulated += avg_cost_np[0]
        count += 1
    return accumulated / count
```

构建训练主循环并开始训练

我们根据上面定义的训练循环数（ `PASS_NUM` ）和一些别的参数，来进行训练循环，并且每次循环都进行一次测试，当测试结果足够好时退出训练并保存训练好的参数。

```
# Specify the directory path to save the parameters
params_dirname = "recommender_system.inference.model"

from paddle.utils.plot import Ploter
train_prompt = "Train cost"
test_prompt = "Test cost"

plot_cost = Ploter(train_prompt, test_prompt)

def train_loop():
    feed_list = [
        main_program.global_block().var(var_name) for var_name in feed_order
    ]
    feeder = fluid.DataFeeder(feed_list, place)
    exe.run(star_program)

    for pass_id in range(PASS_NUM):
        for batch_id, data in enumerate(train_reader()):
            # train a mini-batch
            outs = exe.run(program=main_program,
                                feed=feeder.feed(data),
                                fetch_list=[avg_cost])
            out = np.array(outs[0])

            # get test avg_cost
            test_avg_cost = train_test(test_program, test_reader)

            plot_cost.append(train_prompt, batch_id, outs[0])
            plot_cost.append(test_prompt, batch_id, test_avg_cost)
            plot_cost.plot()
```

```
                                "user_id", "gender_id", "age_id", "job_id",
                                "movie_id", "category_id", "movie_title"
                            ], [scale_infer], exe)
                    return
                print('EpochID {0}, BatchID {1}, Test Loss {2:0.2}'.format(
                            pass_id + 1, batch_id + 1, float(test_avg_cost)))

                if math.isnan(float(out[0])):
                    sys.exit("got NaN loss, training failed.")
```

开始训练

```
train_loop()
```

应用模型

生成测试数据

使用 create_lod_tensor(data, lod, place) 的API来生成细节层次的张量。 `data` 是一个序列，每个元素是一个索引号的序列。 `lod` 是细节层次的信息，对应于 `data` 。比如，data = [[10, 2, 3], [2, 3]] 意味着它包含两个序列，长度分别是3和2。于是相应地 lod = [[3, 2]]，它表明其包含一层细节信息，意味着 `data` 有两个序列，长度分别是3和2。

在这个预测例子中，我们试着预测用户ID为1的用户对于电影'Hunchback of Notre Dame'的评分

```
infer_movie_id = 783
infer_movie_name = paddle.dataset.movielens.movie_info()[infer_movie_id].title
user_id = fluid.create_lod_tensor([[np.int64(1)]], [[1]], place)
gender_id = fluid.create_lod_tensor([[np.int64(1)]], [[1]], place)
age_id = fluid.create_lod_tensor([[np.int64(0)]], [[1]], place)
job_id = fluid.create_lod_tensor([[np.int64(10)]], [[1]], place)
movie_id = fluid.create_lod_tensor([[np.int64(783)]], [[1]], place) # Hunchback of Notre Dame
category_id = fluid.create_lod_tensor([np.array([10, 8, 9], dtype='int64')], [[3]], place) # Ani
movie_title = fluid.create_lod_tensor([np.array([1069, 4140, 2923, 710, 988], dtype='int64')], [
                                      place) # 'hunchback','of','notre','dame','the'
```

构建预测过程并测试

与训练过程类似，我们需要构建一个预测过程。其中， `params_dirname` 是之前用来存放训练过程中的各个参数的地址。

```
place = fluid.CUDAPlace(0) if use_cuda else fluid.CPUPlace()
exe = fluid.Executor(place)

inference_scope = fluid.core.Scope()
```

测试

现在我们可以进行预测了。我们要提供的 `feed_order` 应该和训练过程一致。

```
with fluid.scope_guard(inference_scope):
    [inferencer, feed_target_names,
    fetch_targets] = fluid.io.load_inference_model(params_dirname, exe)

    results = exe.run(inferencer,
                      feed={
                          'user_id': user_id,
                          'gender_id': gender_id,
                          'age_id': age_id,
                          'job_id': job_id,
                          'movie_id': movie_id,
                          'category_id': category_id,
                          'movie_title': movie_title
                      },
                      fetch_list=fetch_targets,
                      return_numpy=False)
    predict_rating = np.array(results[0])
```

总结

本章介绍了传统的个性化推荐系统方法和YouTube的深度神经网络个性化推荐系统，并以电影推荐为例，使用PaddlePaddle训练了一个个性化推荐神经网络模型。个性化推荐系统几乎涵盖了电商系统、社交网络、广告推荐、搜索引擎等领域的方方面面，而在图像处理、自然语言处理等领域已经发挥重要作用的深度学习技术，也将会在个性化推荐系统领域大放异彩。

参考文献

1. P. Resnick, N. Iacovou, etc. “GroupLens: An Open Architecture for Collaborative Filtering of Netnews”, Proceedings of ACM Conference on Computer Supported Cooperative Work, CSCW 1994. pp.175-186.
2. Sarwar, Badrul, et al. "Item-based collaborative filtering recommendation algorithms." *Proceedings of the 10th international conference on World Wide Web*. ACM, 2001.
3. Kautz, Henry, Bart Selman, and Mehul Shah. "Referral Web: combining social networks and collaborative filtering." Communications of the ACM 40.3 (1997): 63-65. APA
4. Peter Brusilovsky (2007). *The Adaptive Web*. p. 325.
5. Robin Burke , Hybrid Web Recommender Systems, pp. 377-408, The Adaptive Web, Peter Brusilovsky, Alfred Kobsa, Wolfgang Nejdl (Ed.), Lecture Notes in Computer Science, Springer-Verlag, Berlin, Germany, Lecture Notes in Computer Science, Vol. 4321, May 2007, 978-3-540-72078-2.
6. Yuan, Jianbo, et al. "Solving Cold-Start Problem in Large-scale Recommendation Engines: A Deep Learning Approach." *arXiv preprint arXiv:1611.05480* (2016).
7. Covington P, Adams J, Sargin E. Deep neural networks for youtube recommendations[C]//Proceedings of the 10th ACM Conference on Recommender Systems. ACM, 2016: 191-198.