**一、下载数据集**

MovieLens 1M 数据集，包含6000个用户在近4000部电影上的1亿条评论。

**1、用户数据(users.dat)**

1::F::1::10::48067

2::M::56::16::70072

3::M::25::15::55117

数据中的格式：UserID::Gender::Age::Occupation::Zip-code

分别对应： 用户ID 性别 年龄 职业ID 邮编

性别："M" for male and "F" for female

年龄： 1: "Under 18"   18: "18-24"   25: "25-34"   35: "35-44"   45: "45-49"

50: "50-55"   56: "56+"

职业：0: "other" or not specified —— 1: "academic/educator" —— 2: "artist"

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"

**2、电影数据(movies.dat)**

1::Toy Story (1995)::Animation|Children's|Comedy

2::Jumanji (1995)::Adventure|Children's|Fantasy

3::Grumpier Old Men (1995)::Comedy|Romance

数据中的格式：MovieID::Title::Genres

分别对应：     电影ID 电影名 电影风格

电影风格有：Action、Adventure、Animation、Children's、Comedy、Crime、Documentary、Dram、Fantasy、Film-Noir、Horror、Musical、Mystery、 Romance、Sci-Fi、Thriller、War、Western  共18种风格

**3、评分数据(ratings.dat)**

1::1193::5::978300760

1::661::3::978302109

1::914::3::978301968

数据中的格式：UserID::MovieID::Rating::Timestamp

分别对应： 用户ID 电影ID 评分 时间戳

UserIDs range between 1 and 6040

MovieIDs range between 1 and 3952

Ratings are made on a 5-star scale (whole-star ratings only)

Timestamp is represented in seconds since the epoch as returned by time(2)

Each user has at least 20 ratings

**二、处理数据**

**分析数据：**

UserID、Occupation和MovieID不用变。

Gender字段：需要将‘F’和‘M’转换成0和1。

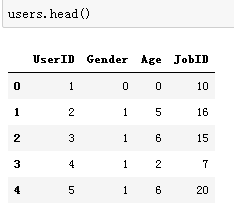
Age字段：要转成7个连续数字0~6。

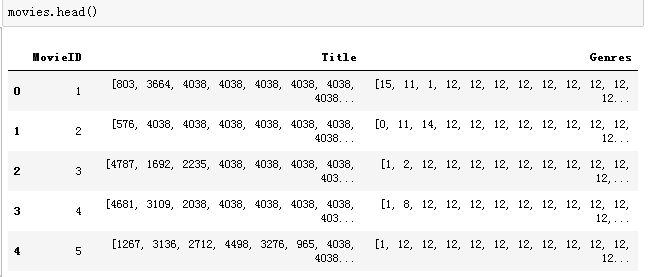
Genres字段：是分类字段，要转成数字。首先将Genres中的类别转成字符串到数字的字典，然后再将每个电影的Genres字段转成数字列表，因为有些电影是多个Genres的组合。

Title字段：处理方式跟Genres字段一样，首先创建文本到数字的字典，然后将Title中的描述转成数字的列表。另外Title中的年份也需要去掉。

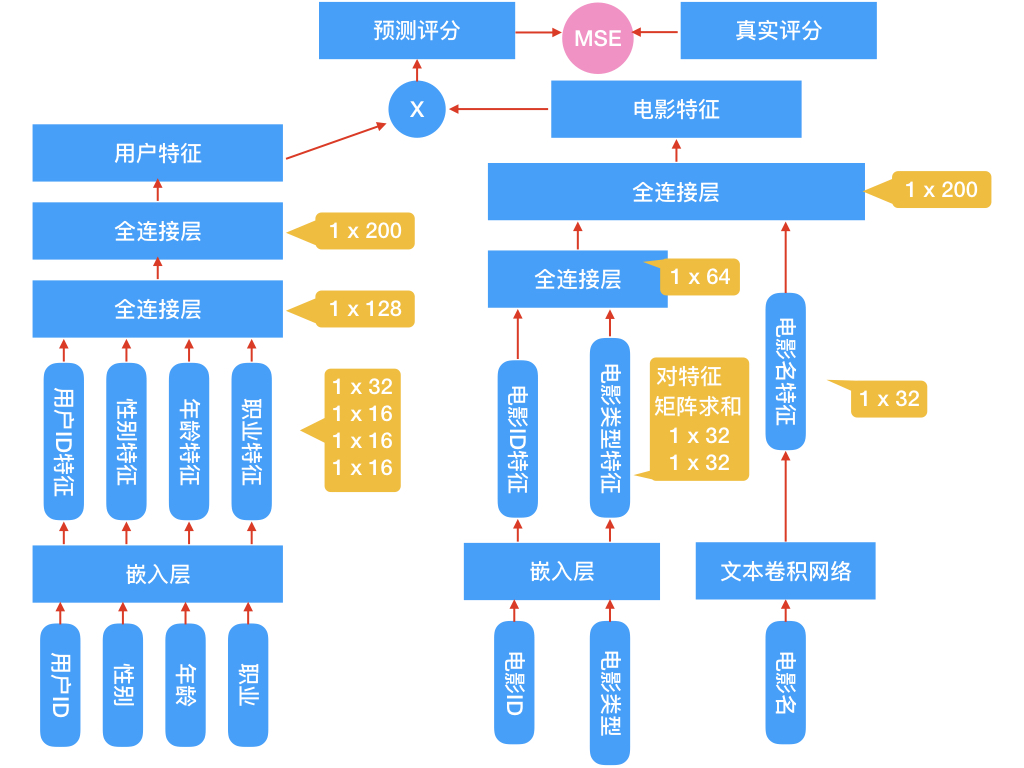
Genres和Title字段需要将长度统一，这样在神经网络中方便处理。空白部分用‘< PAD >’对应的数字填充。

**数据处理后的结果：**





**模型设计**

****

通过研究数据集中的字段类型，我们发现有一些是类别字段，通常的处理是将这些字段转成one hot编码，但是像UserID、MovieID这样的字段就会变成非常的稀疏，输入的维度急剧膨胀，这是我们不愿意见到的，毕竟我这小笔记本不像大厂动辄能处理数以亿计维度的输入：）

所以在预处理数据时将这些字段转成了数字，我们用这个数字当做嵌入矩阵的索引，在网络的第一层使用了嵌入层，维度是（N，32）和（N，16）。

电影类型的处理要多一步，有时一个电影有多个电影类型，这样从嵌入矩阵索引出来是一个（n，32）的矩阵，因为有多个类型嘛，我们要将这个矩阵求和，变成（1，32）的向量。

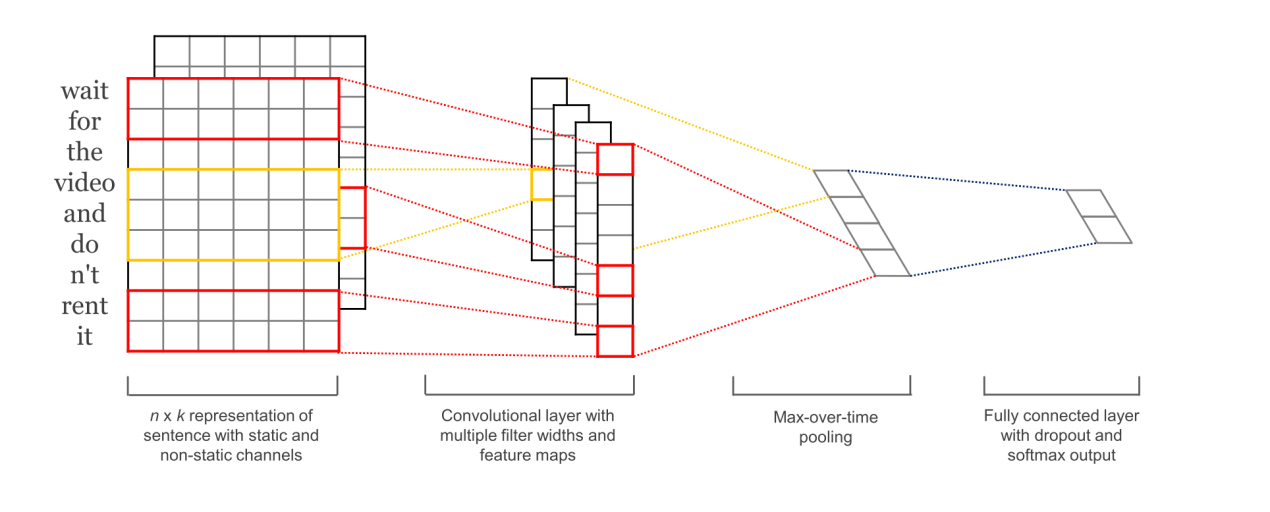
电影名的处理比较特殊，没有使用循环神经网络，而是用了文本卷积网络，下文会进行说明。

从嵌入层索引出特征以后，将各特征传入全连接层，将输出再次传入全连接层，最终分别得到（1，200）的用户特征和电影特征两个特征向量。

我们的目的就是要训练出用户特征和电影特征，在实现推荐功能时使用。得到这两个特征以后，就可以选择任意的方式来拟合评分了。我使用了两种方式，一个是上图中画出的将两个特征做向量乘法，将结果与真实评分做回归，采用MSE优化损失。因为本质上这是一个回归问题，另一种方式是，将两个特征作为输入，再次传入全连接层，输出一个值，将输出值回归到真实评分，采用MSE优化损失。

实际上第二个方式的MSE loss在0.8附近，第一个方式在1附近，5次迭代的结果。

**文本卷积网络**

****

网络的第一层是词嵌入层，由每一个单词的嵌入向量组成的嵌入矩阵。下一层使用多个不同尺寸（窗口大小）的卷积核在嵌入矩阵上做卷积，窗口大小指的是每次卷积覆盖几个单词。这里跟对图像做卷积不太一样，图像的卷积通常用2x2、3x3、5x5之类的尺寸，而文本卷积要覆盖整个单词的嵌入向量，所以尺寸是（单词数，向量维度），比如每次滑动3个，4个或者5个单词。第三层网络是max pooling得到一个长向量，最后使用dropout做正则化，最终得到了电影Title的特征。

**文件顺序：**

1. **download\_data.py 下载数据集**
2. **preprocessing\_data.py 数据预处理**
3. **movie\_inference.py 构建神经网络**
4. **movie\_train.py 训练神经网络**
5. **movie\_save.py 保存训练好的特征**
6. **movie\_recommender.py 开始推荐**

**参考：**

**http://blog.csdn.net/chengcheng1394/article/details/78820529**