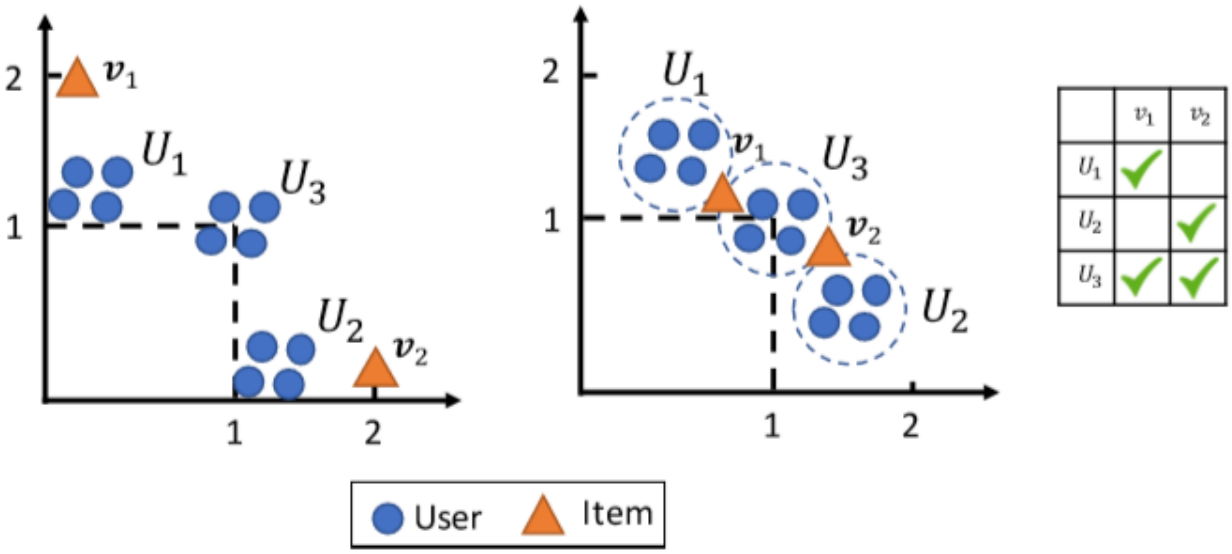
MF(Matrix Factorization)

传统的Matrix Factorization（MF）的方法只能学习到user-item的潜在关系而无法学习到user-user和item-item的Similarity(在最早的Collaborative Filter(CF)是通过基于启发式的计算用户的相似度或者用户的相似度来进行推荐功能，相似度的评判的方式有：欧几里得距离、余弦相似度、皮尔逊相似度等)。

但是近年来以后，Collaborative Filter以矩阵的分解（MF）为主。原始的矩阵分解通过去映射user和item到一个隐因子空间而建模用户的显示反馈，这样user-item的关系就可以通过他们之间的点积所捕获。

虽然矩阵分解取得了一系列的成就，但是它存在一个问题就是矩阵分解（MF）不满足三角不等式，表示存在限制，导师我们的模型是次优解。内积a·b=|a||b|cosθ，也就是它只关注两个向量的大小和角度，也就是它只是在大小和角度的方面来考虑两个向量之间相似度。使用点乘的大小来表示两个向量的距离，点乘越大，就代表两个向量越近，值越小就代表距离越远。这里可以看成距离（也就是相似度）不具有传递性。举例来说：就是如果X与Z都相近，那么Y和Z也应该相近。



CML (Collaborative Metric Learning)

通过学习一个数据集（user-item的数据对，并且是正相关的），通过学习user-item之间的度量，拉近数据集中正相关的item（这里只是相对的），推远其他的item，可以得到结果。

1. 共同喜欢相同物品的用户在一起。
2. 同一用户共同喜欢的item在一起。

最后，任何给定的用户的近邻变成：该用户之前喜欢的item，以及与该用户具有相似品味的用户喜欢的item

CML的损失函数一共有三个：

1. 距离损失函数

度量使用的是欧几里得距离,其中ui为一个useri的隐向量，vj为一个itemj的隐向量

$$d(i,j) = \|\mathbf{u}_i - \mathbf{v}_j\|,$$

只采用LMNN中的push操作，没有使用pull操作（一个item可以被很多的user喜欢，所以不能最小化pull），矩阵分解的本来是使用的均方误差，使用的是打分减去对应的user向量和item向量的乘积在加上正则化项。这里采用的是Bayesian Personalized Ranking（贝叶斯个性化排名）和WARP（Weighted Approximate-Rank Pairwise的结合。

其中贝叶斯个性化排名将明确的评级转化为建模item之间的相对偏好，其中贝叶斯个性化排名中采用的的是点乘的大小来衡量相似度，这里采用欧几里得距离来衡量useri与itemj之间的关联性。

$$\mathcal{L}_m(d) = \sum_{(i,j)\in\mathcal{S}} \sum_{(i,k)\notin\mathcal{S}} w_{ij}[m + d(i,j)^2 - d(i,k)^2]_+, \quad (1)$$

其中S为数据中具有正相关的集合，即itemj为用户i喜欢的，itemk为用户i不喜欢的。这里的[Z]+=max(z,0)为hinge loss函数，wijwij叫做ranking loss weight，后面会提到，m>0为安全边界值。

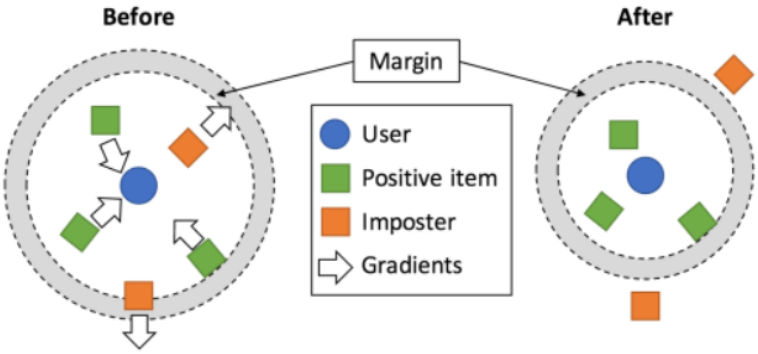


Figure 1: An illustration of collaborative metric learning. The hinge loss defined in Eq. 1 creates a gradient that *pulls* positive items closer to the user and *pushes* the intruding impostor items (i.e., items that the user did not like) away until they are beyond the safety margin.

可以看到在loss函数的控制下,正相关的item与user的距离沿着梯度缩小,而不相关的则相反。其中impostor表示在安全边界内出现的负样本(与user不相关的item)。

加权的排名损失（**Approximated Ranking Loss**）

距离损失函数中的wijwij采取一种叫做WARP（Weighted Approximate-Rank Pairwise）的loss来建立

$$w_{ij} = log(rank_d(i,j) + 1).$$

目的是来惩罚排名靠后的正相关item项。其中: rankd(i,j)为useri的推荐列表中itemj的排名,通过以下的策略来近似: 1、对于每个user-item对(i,j)，采样U个负样本item，并计算距离损失函数。损失函数非0即为impostor。

2、让M代表U中impostor的个数，则其中rankd(i,j)可近似为：⌊J*M/U⌋其中，J为所有的item数。

2. 特征损失函数

本文借助隐式反馈作为用户喜欢的item的feature，f(x)将特征x映射到user-item空间的函数(即为本文的学习结果),vj为相应的item向量,特征损失函数的目的是把item的特征向量作为item本身的一种高斯先验，来调整vj的位置,即相同feature的item应该离得更近,并且能改善那些很少评分的item的位置。f(x)由训练得来,使用MLP + dropout实现。 特征损失函数如下:

$$\mathcal{L}_f(\theta, \mathbf{v}_*) = \sum_j \| f(\mathbf{x_j}, \theta) - \mathbf{v}_j \|^2.$$

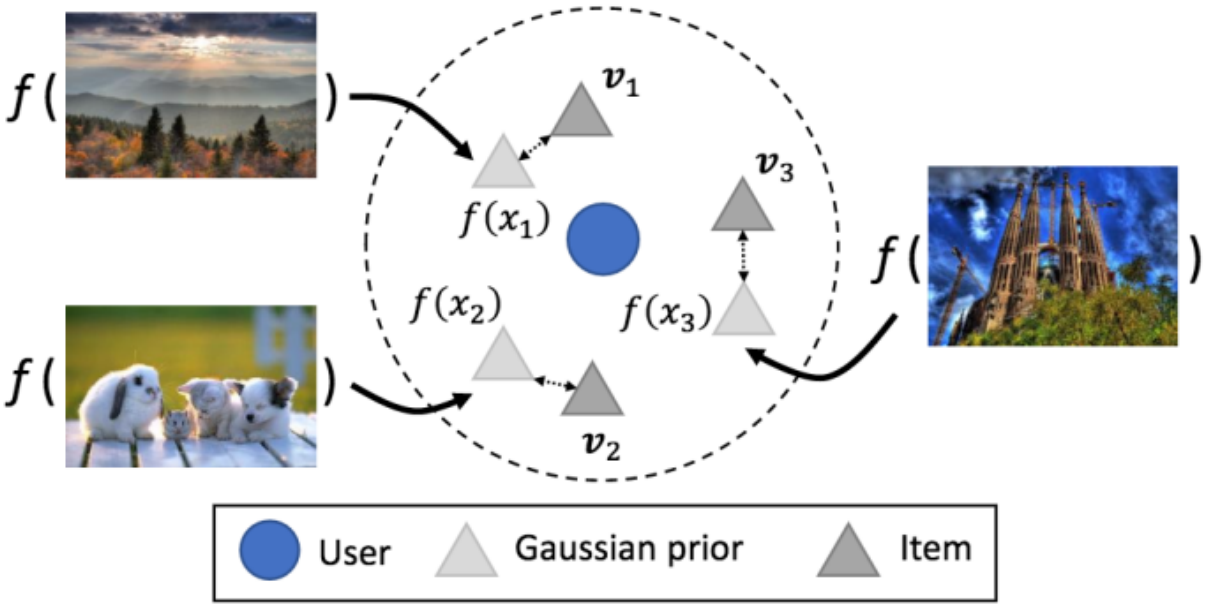


Figure 2: A learnable transformation function f is used to project item features (e.g., image pixels) into the user-item joint space. The projections are treated as a Gaussian prior for items' locations.

上面说到f(x)是一个特征函数,负责将item的特征映射到user-item Space中。我们在这里将其看成是item本身的先验来后去调整vj的位置,使得相同特征的item离得更近。作者使用2层MLP+0.5 Dropout,256-D Hidden-layer. 这个特征函数f(x)是跟我们最后的目标函数一起训练的。我们可以理解成有很多item其实有很多是具有相同特征的(如一篇文章的tags等),通过一个NN将提取出的item特征映射到user-item空间中作为item的先验,可以使得具有相同特征的item,vj在Space尽可能离得更近(因为具有相同特征的vj随机初始化的vector可能相差很大,而提取出的相同特征经f(x)映射到Space位置大体相差不大,所以可以起到调节作用)。另外对于那些和user交互很少的item,他们本身就无法根据三角形不等式使item之间的位置靠的很近,通过特征函数的映射来调整他们之间的相对位置也是一个很不错的办法。同时这里的f(x)还可以解决RS的cold-start问题,就是说因为我们已经训练好了f(x)后,它是将item feature 或者user feature映射到user-item Space的,如果我们现在有一个新user他没有任何的implicit feedback,我们可以通过f将其映射到user-item space然后预测其Top-n。同理一个新item也可以这样添加进去。当然,对于user我们也可以使用该方法。

3. 正则项损失函数

$$\mathcal{L}_c = \frac{1}{N}(\|C\|_f - \|diag(C)\|_2^2),$$

其中C为协方差矩阵:

$$C_{ij} = \frac{1}{N}\sum_n (y_i^n - \mu_i)(y_j^n - \mu_j),$$

其中

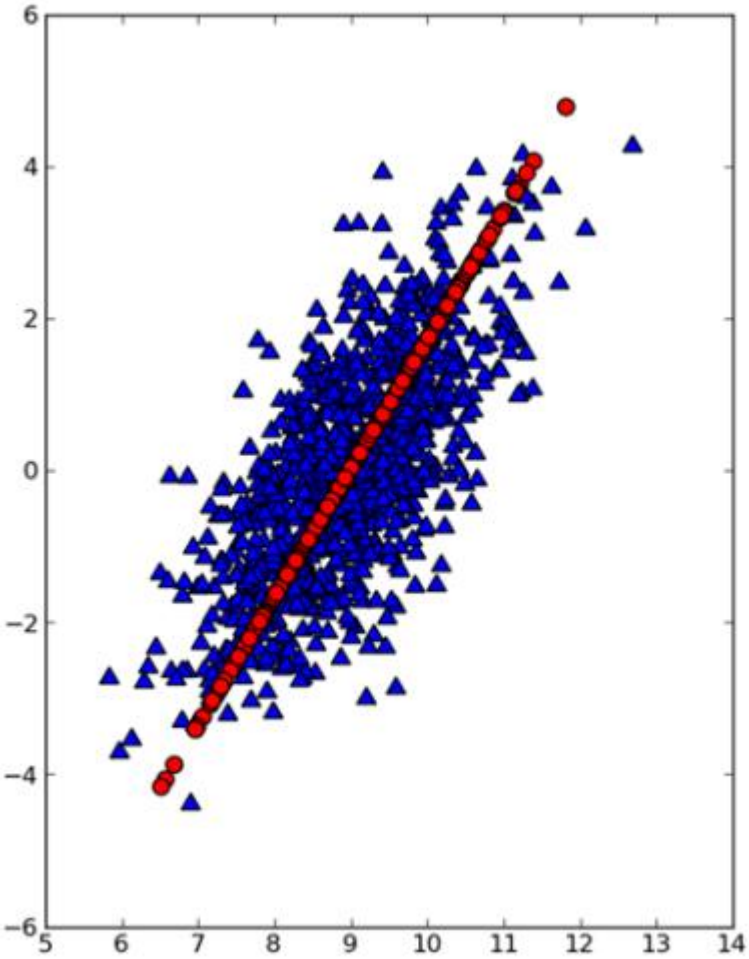
$$y_i^n$$

为user or item vector,n为batch中的索引,i为向量的维度。

$$\mu_i = \frac{1}{N}\Sigma_n y_i^n$$

N为batch大的大小。协方差正则的引入,主要是解除各维度间的相关性。因为协方差可以看成向量间的冗余,这样能使得整个空间能进行更多信息的表达

正则损失函数主要是为了减少过拟合。和Dropout思想类似，通过减少各隐藏层神经元之间的相关性去减少过拟合。该文章中也使用了该正则化方法,。其实我觉得这里的思想和PCA里的降维很类似(也是根据协方差矩阵来减少各维度之间的相关性,但是我们这里不降维)。我们知道这里我们需要去将每一个user,item embedding到user-item空间中,我们假设他是二维的,这样显得更直观一些。如图



如果我们的user和item是这样的,那么我们就可以发现我们造成了维度浪费,因为图中红线就可以表示他们了(也就是说我们选了r-Dimension其实根本不用),同时可以发现通过协方差接触各维度相关性的同时可以使得我们点”相对分散”,去利用好我们的整个Space.

训练过程

目标函数:

$$\min_{\theta,\mathbf{u}_*,\mathbf{v}_*} \mathcal{L}_m + \lambda_f \mathcal{L}_f + \lambda_c \mathcal{L}_c$$

$$\text{s.t.} \quad \|\mathbf{u}_*\|^2 \leq 1 \text{ and } \|\mathbf{v}_*\|^2 \leq 1,$$

这是总的损失函数，下方的限制条件为保证空间可计算性。整个过程为Mini-Batch SGD训练，学习率采用AdaGrad控制。训练步骤如下：

1. 从S中抽样N个正相关user-item对
2. 对于每一对,抽样U个负样本item,并计算近似的randd(i,j)=⌊J×M/U⌋
3. 对于每一对,保留使得距离损失函数最大的那个负样本item(k),并组成一个batchN。
4. 计算梯度并更新参数
5. 重复此过程直到收敛