推荐系统

**什么是推荐系统？**

YouTube如何知道你下面想看哪一个视频？Google商店如何为你推荐app？在这背后，有一个基于ML的推荐模型决定了视频和app和你喜欢的其他东西有多相似，并提供推荐结果。现实中经常使用两种推荐系统：  
 主页推荐

相关物品推荐

主页推荐

主页推荐是基于已知的用户偏好，为其进行个性化推荐。每个用户看到不同的推荐结果。

相关物品推荐

相关物品推荐是推荐和一个特定物品相似的对象。

**为什么用推荐系统？**

推荐系统帮助用户在大型语料库中找到有吸引力的内容。比如，Google商店提供数百万个app，Youtube提供数十亿个视频。每天会增加更多app和视频。如何使用户找到新的感兴趣的内容。可以使用搜索来获得，然而推荐系统可以在用户不搜索的情况下将这些结果显示出来。

术语

Items

推荐系统推荐的对象。

Query

推荐系统进行推荐使用的信息，Queries可以是以下内容的综合：

User information

User id

User过去看过的item

Additional context

日期时间

用户的设备

Embedding

一个离散集合(the set of query或者the set of items)到一个向量空间的映射叫做embedding space。许多推荐系统依赖于学习一个对queries和items的适当的embedding表达

**推荐系统综述**

常见的推荐系统包括以下组成部分：

Candidate Generation召回、scoring评分、re-ranking重排序

召回

在第一阶段，系统从一个潜在的巨大语料库开始，生成一个更小的候选子集。例如，YouTube中的候选生成器将数十亿个视频减少到数百或数千个。考虑到语料库的巨大规模，该模型需要快速评估查询。给定的模型可以提供多个候选生成器，每个生成器提名不同的候选子集。

评分

另一个模型对候选项进行评分和排序以便选择要显示给用户的item集合，由于此模型在一个较小的集合上计算，因此这个模型可以基于更多的queries提供更精确的结果

排序

最后，系统必须考虑到最终排名的附加限制。例如，系统会删除用户明确不喜欢的项目，或增加更新鲜内容的分数。重新排名也有助于确保多样性、新鲜度和公平性。

我们将在课程中讨论每一个阶段，并给出来自不同推荐系统（如YouTube）的例子。

**召回概述**

召回是推荐的第一步。对于一个query，系统生成一组相关的候选对象。下表显示了两种常见的召回方法：

|  |  |  |
| --- | --- | --- |
| 类型 | 定义 | 举例 |
| 基于内容过滤 | 基于item之间的相似度和用户的喜好来推荐item | 如果A用户看了两个猫视频，然后系统为其推荐猫的视频 |
| 协同过滤 | 同时使用query和items之间的相似度来推荐 | 如果用户A和B相似，且B喜欢视频1，那么将为A推荐视频1，即使A没有看过和A类似的视频 |

Embedding space

基于内容过滤和系统过滤将每一个item和query映射为一个embedding向量，这些向量有同一纬度。通常，embedding space是低维的，并表达了item或query集合的潜在结构。相似的item，比如同一个用户经常观看的不同YouTube视频，在embedding space里的距离相近。这里的距离由相似度来测量。

Similarity measure

相似度测量是一个使用公式E x E->R计算一对embedding向量，并返回一个标量来测量他们之间的相似度。Embeddings可以用来召回：对于一个给定的query embedding Q，系统寻找和Q相近的item embedding X，相似度用s(Q,X)计算。

大多数推荐系统使用以下的相似度：

Cosine

Dot product

Euclidean distance

Cosine

两个embedding向量之间角度的余弦值 s(Q,X) = cos(Q,X)

Dot product

Dot product是两个embedding向量之间的内积。如果这两个向量是单位向量，那么Dot product和Cosine是一样的

Euclidean distance

连个embedding向量之间的欧式距离，距离越小意味着相似度越大，注意当向量为单位向量时，向量之间的欧式距离和Cosine、Dot product的和为1，即为1-cos(Q,X)

选择哪一个相似度度量？

和cosine相比，dot product还受到向量模的影响。就是说，向量的模越大，相似度越大，越容易被推荐系统推荐，这样推荐系统容易受到以下因素的影响：

训练集中越常出现的item的embedding向量具有越大的模。如果希望获得item热度信息，那么可以选择dot product。然而，如果对这种现象不采取任何措施，那么推荐结果将会被热门item霸占。实际情况下，你可以使用相似度计算的变形来降低item embedding向量模的影响。比如：

很少出现的item在训练中不会经常出现，因此，如果它们初始化具有一个很大的模，那么推荐系统会优先推荐这些稀有item。因此，要避免这样的问题，需要小心初始化，并适当使用正则化，我们将在第一个练习中进行详细说明。

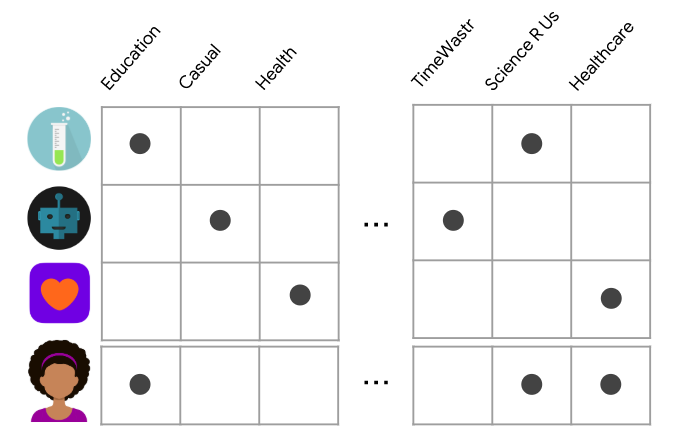
**基于内容过滤**

基于内容过滤根据用户历史操作或显示反馈，推荐和他们喜好相似的item。

为了演示基于内容过滤，让我们为google商店手动设计一些特征。下图展示了一个特征矩阵，每一行代表一个应用，每一列代表一个特征，特征包括分类比如教育、休闲、健康，app发行者等等。为了简化，假设这个特征矩阵是二进制的，即不为0代表应用具有这个特征。

你还可以使用相同的特征空间来表示用户。用户可以显示地提供一些特征。比如，用户在个人资料里选择娱乐类应用。其他特征可以是隐式的，基于用户之前安装的app，比如用户安装了Science R Us发布另一个应用。

推荐模型应该为用户推荐和其相关的item。你必须使用一个相似度矩阵，然后基于这个矩阵建立一个推荐系统来为每一个候选item打分。



使用Dot product作为Similarity Measure

假设user embedding 和app embedding的元素都是二进制的。那么dot product就是两个向量共同具有的特征数。共有特征数越多，dot product越大，相似度越大。

优势

这个模型不需要任何user数据，因此它很容易扩展到很多的user

这个模型可以捕捉用户的特别兴趣，能够为用户推荐小众的但是很合适的item

劣势

因为item的特征表达在一定程度上由人工维护，因此这个方法需要许多专业知识。所以，手工维护的特征影响模型的效果

这个模型只能基于用户现在的兴趣进行推荐，换句话说，这个模型不会扩展用户现有的兴趣。

**协同过滤**

为了解决基于内容过滤的限制，协同过滤基于user和query同时的相似度来提供预测。这允许偶然性推荐，即协同过滤基于和A用户相似的B用户的兴趣，为A进行推荐。另外，embedding可以自动学习，不需要人工维护特征。

电影推荐案例

考虑一个电影推荐系统，训练数据由一个反馈矩阵构成：

每一行代表一个user

每一列代表一个item movie

电影反馈数据包含两种：

显示的-用户给电影评分，标明对电影的喜好成都

隐式的-用户看过一个电影，系统推断这个用户对这个电影感兴趣

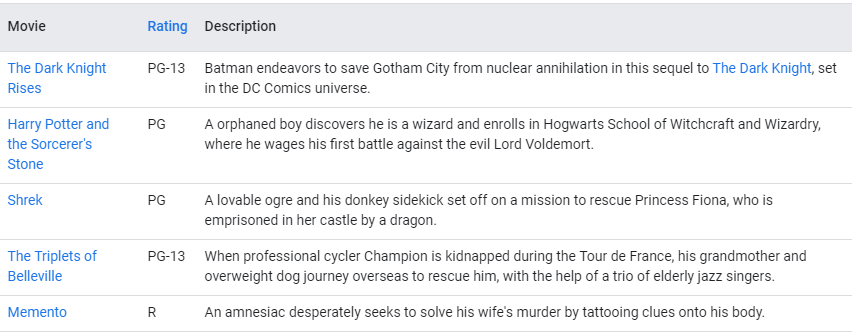
为了简化，我们将假设反馈矩阵是二进制的，即1表示用户喜欢这个电影。

当一个用户浏览主页，系统将依据以下内容为其推荐：

用户历史喜好的相似电影

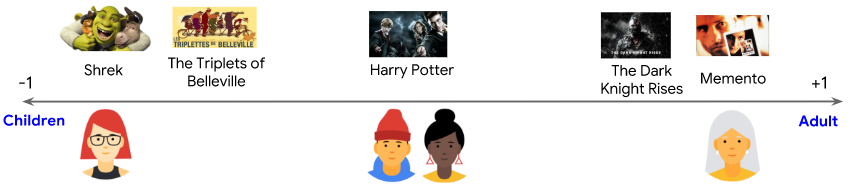
相似用户喜好的电影

为了便于表达，我们人工选定一些电影特征如下：

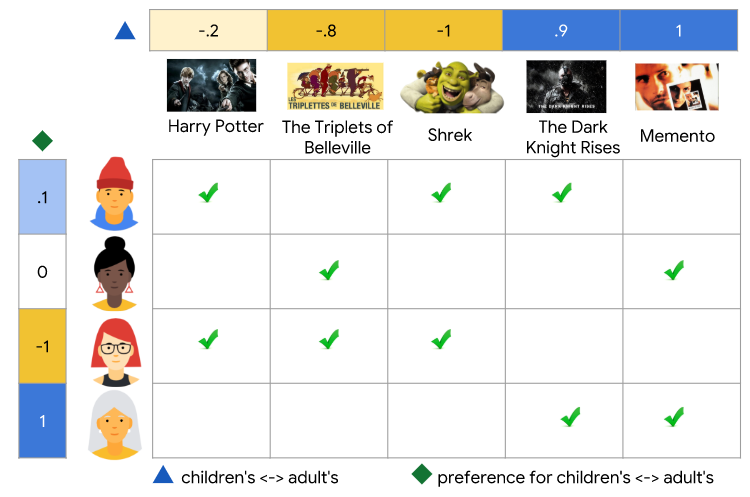


1-D embedding

假设我们给每一个电影打一个-1到1之间的分数，这个分数表明合格电影适合儿童还是成年。假设我们还给每一个用户打一个-1到1之间的分数，用来表明在这个用户喜欢儿童电影还是成年电影。

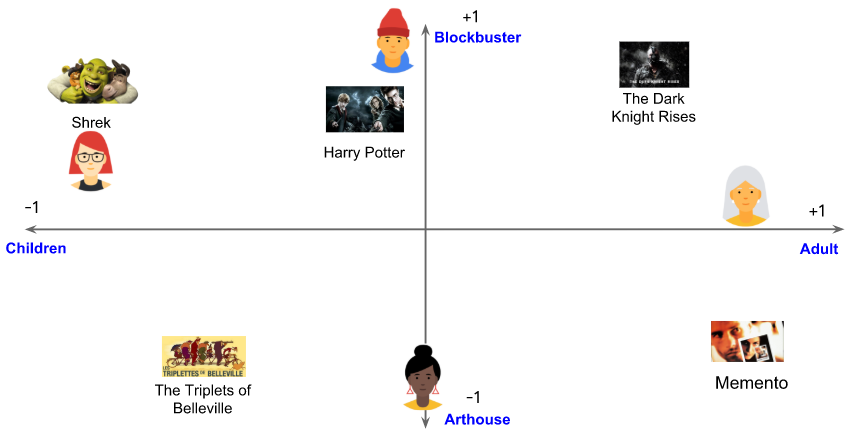


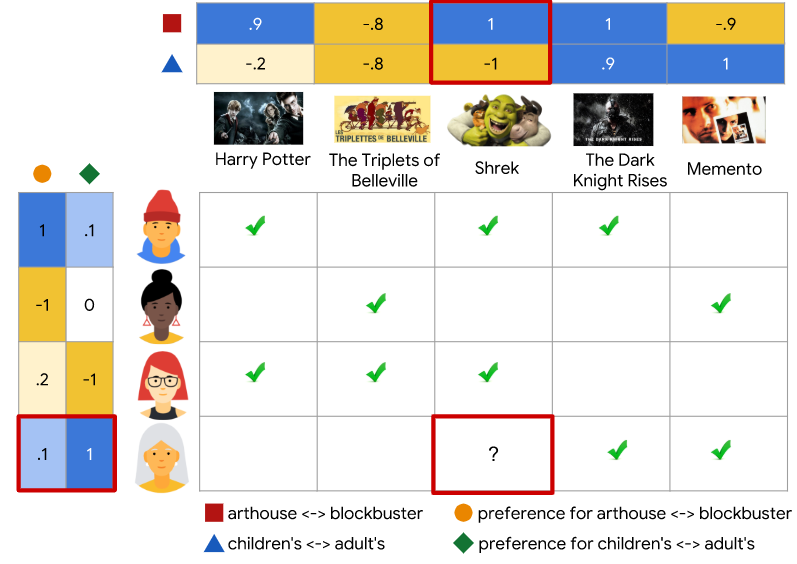
下图中，对勾表示用户实际观看的电影。第三个和第四个用户的偏好能够很好地表示，但是第一和第二个用户的偏好不明显。



2D Embedding

一个特征不能用来解释用户的偏好，我们需要加入第二个特征，电影是娱乐片还是艺术片。这样我们可以用二维embedding来表示电影和用户





在这个例子中，我们使用人工特征来进行embedding，实际上，这些embedding可以自动学习得到。在后面的两个部分，我们将讨论不同的模型来学习这些embedding，以及如何训练他们。

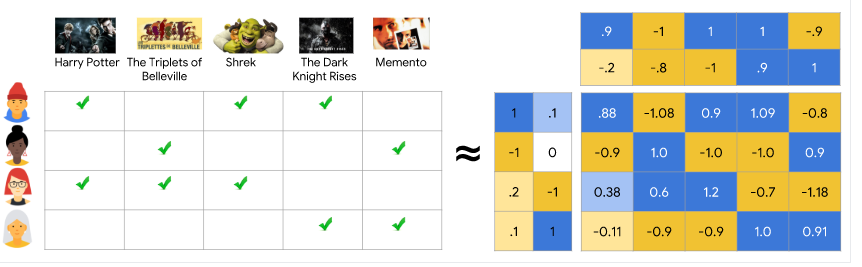
这个方法的协同特点就体现在模型学习这些embedding的过程中。假如电影的embedding vector是固定的，那么模型将学习到用户的embedding向量来解释他们的偏好，且具有相似偏好用户的embedding向量距离接近。同样的，如果用户的embedding 向量是固定的，我们可以学习到电影的embedding向量来解释这样一个反馈矩阵，结果相似用户喜欢的电影的embedding向量距离接近。

Matrix Factorization

矩阵分解是一个简单的embedding模型。对于给定的反馈矩阵A ∈ Rmxn，m是user或queries的数量，n是item数量，这个模型学习的embedding向量为：

一个user embedding矩阵U∈Rmxd, 第i行表示user i的embedding

一个item embedding矩阵V∈Rnxd,第j行表示item j的embedding



学习到的U和V，UVT是矩阵A的近似。

选择目标函数

一个直观的描述UVT和A相似程度的目标函数是平方距离：



在这个目标函数中，只是累加了所有观察到的(i,j)的误差，也就是说反馈矩阵中所有为1的值。然而只计算这些值是一个不好的主意，因为这样会得到一个很小的损失，并产生一个预测结果不好的模型。

我们可以把没有观察到的值也考虑进来，这样就是计算矩阵A和估计矩阵UVT之间squared Frobenius distance的最小值

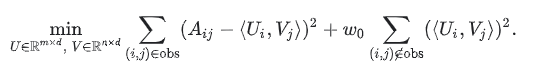


你可以使用矩阵A的SVD来解决这个二次问题，然而SVD并不是一个很好的解决方法，因为在实际的应用中，矩阵A是非常稀疏的。比如考虑YouTube的用户观看电影行为，UVT将非常接近与零矩阵，从而导致非常差的表现。

Weighted Matrix Factorization将目标函数分解成两个部分：

观测值得总和

非观测值综合



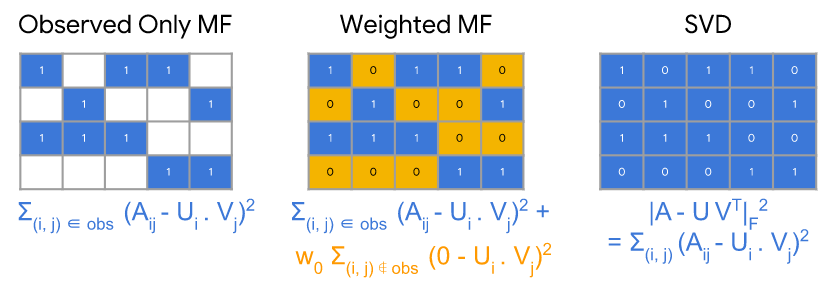
这里w0是一个超参数，用来调整这两部分得权重，避免某一部分控制目标函数，调整w0

非常重要。

在实际应用中，你也需要对观测值进行权重调整，否则，经常出现得items和行为频繁得user会控制目标函数。你可以通过给训练样本加上权重来减轻这个影响。这样目标函数就变为



Wi,j是i和j共现频率得函数。



最小化目标函数

常见得目标函数最小化算法包括：

SGD随机梯度下降是一个最小化损失函数得通常方法，

Weighted Alternating Least Squares是针对这个问题得特定算法

目标函数是二次的，WALS首先随机初始化embedding，即U和V，然后进行交替迭代操作：

固定U，更新V

固定V，更新U

每一个步骤都有准确解，因为此时目标函数是线性的。这个方法能保证收敛，因为每一步都能保证损失在降低。

SGD和WALS

这两种方法优劣对比：

SGD

优势 非常灵活

优势 可以并行

劣势 收敛速度慢

劣势 对于没有观察到的样本不好处理 需要使用负采样

WALS

劣势 只依赖于平方损失

优势 可以并行

优势 收敛速度比SGD快

优势 可以很好地处理没有观察到地样本

协同过滤的优势和劣势

优势

不需要专业知识

不需要人工选择特征

偶然性

可以帮助user扩展新兴趣

很好的起点

在某种程度上，协同过滤只需要反馈矩阵就可以用来训练矩阵分解模型，不需要其

他上下文特征。实际上，协同过滤可以作为多个候选生成器之一（召回源）

劣势

不能够处理新的item

模型预测是使用给定的user、item的embedding，计算dot product。如果有一个

新的item在训练集中没有出现过，那么模型将不能为这个item创建embedding，

这就是冷启动问题。然而可以通过以下方法来在一定程度上解决这个问题。

**Projection in WALS** 对于一个没有在训练集中出现过的item i0，如果系统中有一些用户和这个item的交互，那么可以计算出这个item的Ui0，系统只需求解以下方程或者加权版本：



计算过程可以看作ALS中的一步，即固定user embedding，更新V。对于一个新的用户，过程类似。

**Heuristics to generate embeddings of fresh items** 如果系统没有交互，系统可以通过从相同的类别、来自相同的上传者（在YouTube）中的项的平均化来近似其embedding，等等。

不能包括query或item的其他特征

其他特征是指除了query或item ID之外的特征，比如国家、年龄等。包含这些特

征能够提升模型质量，WALS中不容易添加这些特征，但是在扩展的WALS中可以

做到。

扩展WALS省略

**深度神经网络模型**

前面介绍了如何使用Matrix Factorization来学习embedding。MF的局限包括：

只使用了query或item的ID作为特征，这样导致冷启动问题

推荐的相关性，如在colab的例子中所见，流行的item总是推给每一个user，特别是

使用Dot product作为相似度度量的时候。最好能捕捉到用户的特定兴趣。

DNN模型可以解决MF的局限。DNN能够和query特征、item特征结合（复杂的输入层），这对捕捉到一个user特定兴趣并进行相关推荐是有帮助的。

Softmax DNN for recommendation

一个可能的DNN模型是softmax，一个将问题看作多分类预测的方法：

输入是user query

输出是一个概率向量，其大小等于语料库中的项目数，表示与每个项目交互的概率；例

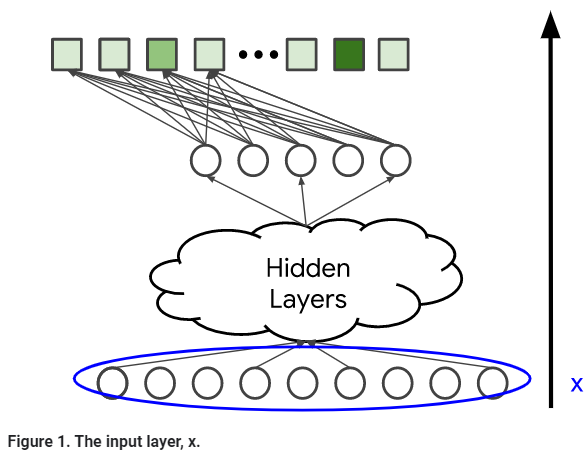
如，单击或观看YouTube视频的概率。

Input

DNN的输入包括：

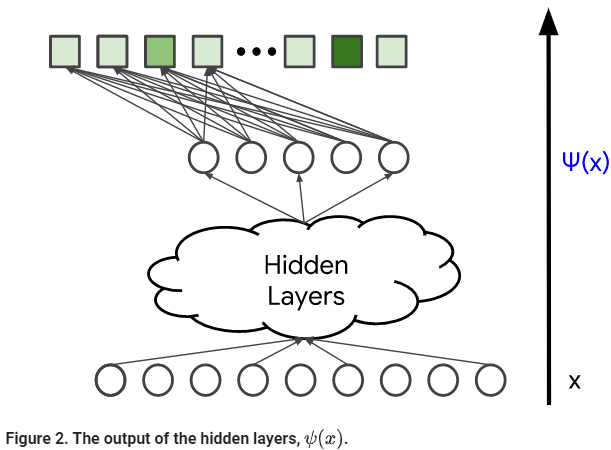
dense feature，比如观看时间，最后一次观看时间

sparse feature，比如观看历史和国家



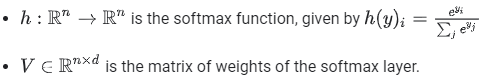
模型结构

模型结构决定了模型的复杂度和表现能力。通过增加隐藏层和非线性激活函数，比如relu，模型可以捕捉数据之间更复杂的关系。然而参数数量增加也会使模型更难训练，以及提供服务更昂贵。我们使用ψ(x)表示最后一层隐藏层的输出。

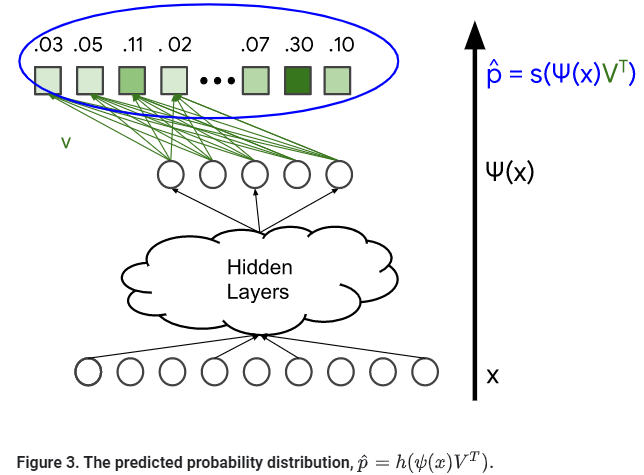


Softmax Output：predicted probability distribution

模型输出层ψ(x)通过softmax层映射为概率分布：

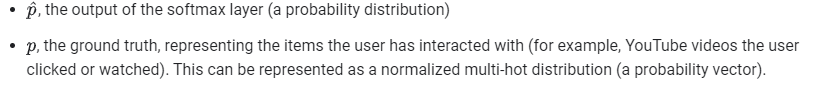


Softmax层将一个向量y∈Rn，通常是log，映射为概率分布

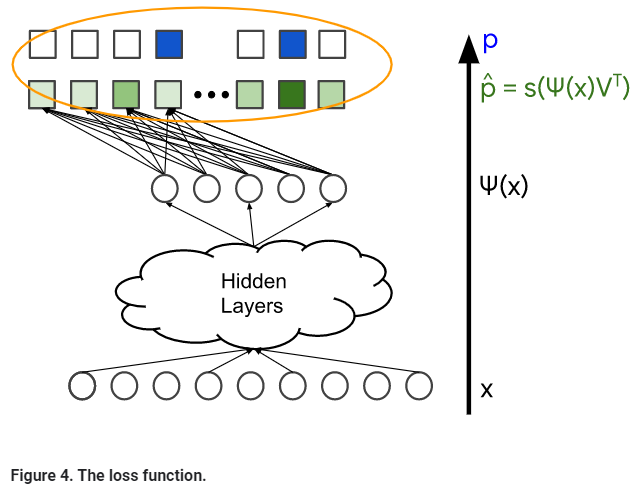


Loss Function

定义一个损失函数：

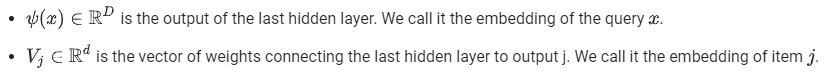


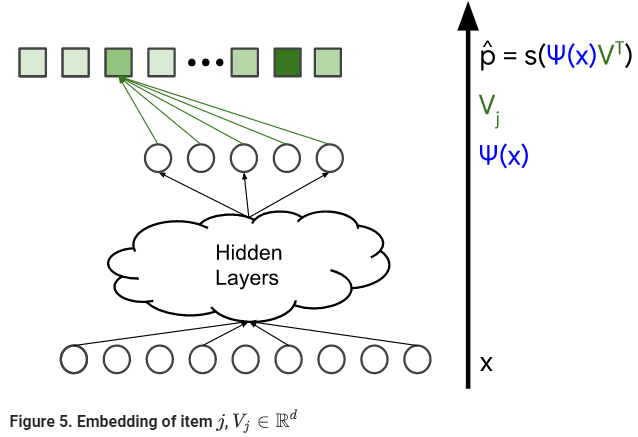
比如，你可以使用交叉熵损失因为你在对比两个概率分布



Softmax Embeddings

User观看Item j的概率为这里Z是一个不依赖于j的标准化的常数。换句话说，，所以item j的log概率是两个D维向量的dot product，这两个向量可以解释为query和item embedding：





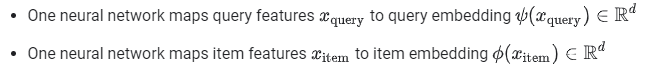
DNN和MF

在softmax和MF模型，系统对每一个item j都学习了一个embedding Vj。我们在MF中叫做V的item embedding矩阵，对应softmax层的权重矩阵V。

但是query embedding是不同的。MF对每一个query i学习一个Ui，而softmax学习一个从query feature x到一个embedding ψ(x)的映射。所以你可以将DNN看作一个扩展的MF，不过使用非线性函数ψ(x)代替了原来的query。

在DNN中使用Item feature

可以将上述方法用在item上么？就是不在为每一个item学习一个embedding，而是让模型学习一个item特征到非线性函数的映射？答案是肯定的，我们可以使用一个双塔神经网络，包括两个神经网络：



模型的输出可以看作两个非线性函数的dot product 。注意这里没有softmax层了，对于每一个query，新的模型预测一个值(Xquery,Xitem)，而不是一个概率向量。

Softmax training

Negative sampling

由于损失函数需要对比两个概率向量，如果样本数量过大，那么计算loss梯度将会非常昂贵。

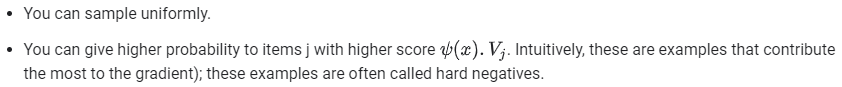
你可以只是用正样本你来计算梯度，但是如果系统只使用正样本训练，模型将会被folding。

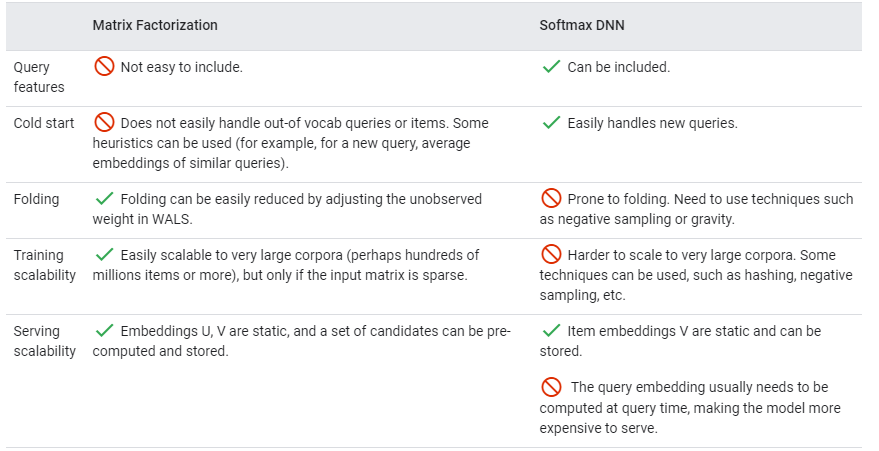
不使用全部数据进行训练，也不能只用正样本，我们可以采用负采样negative sampling。

即使用下面的样本来计算近似的梯度：



负采样的策略有：





矩阵分解通常是大型语料库的较好选择。它更容易扩展，查询更便宜，也不容易folding。

DNN模型可以更好地获取个性化的偏好，但是训练难度更大，查询成本更高。DNN模型比矩阵分解更适合评分，因为DNN模型可以使用更多的特征来更好地捕捉相关性。此外，DNN模型折叠通常是可以接受的，因为您主要关心的是对假定相关的预筛选候选集进行排序。