大规模图推荐模型的快速优化机制

杨正一[1], 吴 宾[2], 王 翔[3], 何向南[1, 4]

[1] 中国科学技术大学 信息科学技术学院 合肥 中国 230026
[2] 郑州大学 信息工程学院 郑州 中国 450001
[3] 新加坡国立大学 NExT++中心 新加坡 新加坡 119077
[4] 中国科学技术大学 大数据学院 合肥 中国 230026

摘要 推荐系统在帮助用户从海量数据中发现自己感兴趣的信息时能起到重要作用。近些年来, 深度学习在计算机视觉等诸多领域卓有成效, 吸引了越来越多推荐系统领域学者的关注。推荐系统结合图神经网络等深度学习方法取得了令人瞩目的效果。然而, 现存的许多方法主要关注在如何用深度学习模型来设计推荐系统的架构, 却少有工作关注推荐系统的优化框架, 尤其是从优化框架方面提升推荐系统的训练效率。因此随着模型的日益复杂, 训练模型的时间代价也越来越大。

本工作中, 我们试图从优化框架方面提升大规模图推荐模型的训练效率。推荐系统中最主流的模型优化框架为贝叶斯个性化排序(Bayesian personalized ranking,BPR), 其潜在假设是目标用户对于已交互的物品的喜好程度强于未交互的物品, 然后通过最大化用户对感兴趣物品和不感兴趣物品的评分差来实现。然而, BPR 优化器的瓶颈在于模型参数的学习效率低下, 在计算资源有限, 且用户的兴趣要具有时效性等现实因素下, 极大限制了主流图推荐模型在工业场景中的应用。究其原因, BPR 优化器需要每个训练样本对单独经过非线性激活函数, 这样元素级别的运算无法转化为矩阵操作等并行计算的形式, 进而未能发挥 GPU 的并行加速性能。受平方误差损失函数在结合推荐任务时, 对矩阵化操作较为友好的启发, 我们设计了一种快速非采样优化器 FGL, 可广泛适用于主流图推荐模型。经过一系列理论推导与转换, FGL 有效规避了损失函数中复杂度较高的计算项, 极大提升了模型的训练效率。以经典矩阵分解模型和最先进的图神经网络模型 LightGCN 为代表, 本文在四个基准数据集上进行了大量的实验。实验结果表明, FGL 优化器在保证推荐准确度下, 其训练效率相比于 BPR 获得了数量级层面的加速, 表明 FGL 在现实工业场景中具有很大的应用潜力。

关键词 推荐系统; 图神经网络; 优化框架; 训练效率
中图法分类号 TP183 DOI 号 10.19363/J.cnki.cn10-1380/tn.2021.09.08

A Faster Optimization Mechanism for Large-scale Graph Recommendation Models

YANG Zhengyi[1], WU Bin[2], WANG Xiang[3], HE Xiangnan[1, 4]

[1] School of Information Science and Technology, University of Science and Technology of China, Hefei 230026, China
[2] School of Information Engineering, Zhengzhou University, Zhengzhou 450001, China
[3] NExT++ Center, National University of Singapore, Singapore 119077, Singapore
[4] School of Data Science, University of Science and Technology of China, Hefei 230026, China

Abstract Recommendation system plays an important role in helping users find their personalized interested information from the overwhelming data. In the recent years, as deep learning has shown its extraordinary performance in a wide variety of fields such as computer vision, it attracts more and more attention from the researchers in the field of recommendation systems. Many deep learning models such as graph neural networks have been applied into recommendation systems and achieved state-of-the-art performance. However, existing methods have mainly focused on exploring how to design the structures of recommendation systems with deep models, while few works have considered the optimization framework and how to improve the efficiency of recommendation system in respect to it. Therefore, as the models become more complex, it takes more and more time to train the models.

In this work, we manage to speed up the training process of large-scale graph recommendation models in respect to the optimization phase. The most often used optimization framework for recommendation system is Bayesian personalized ranking (BPR), which supposes that a targeted user has more preference on items he has interacted with than others, and achieves it by maximizing the margin between the predicted interested scores and the uninterested ones. However, the bottleneck of BPR exists in the inefficiency of model parameter learning, which massively limits industrial applications of mainstream graph recommendation models as the computation resources are limited and users' interest should be updated

通讯作者: 何向南, 博士, 教授, Email: hexn@ustc.edu.cn
本课题得到国家自然科学基金(No. U19A2079, No. 61972372)资助。
收稿日期: 2021-04-30; 修改日期: 2021-08-08; 定稿日期: 2021-08-10

as fast as possible. The fundamental reason lies in the formulation of BPR: every pair of training tuple is transformed by a non-liner activation function separately with only element-wise computation and no matrix or vector operation, which fails to fully leverage GPUs' speed-up ability as GPU is designed for fast parallel computing. Inspired by the fact that square loss is more friendly for matrix operation when it is combined with recommendation tasks, we propose a fast and generic learner: FGL, which has no sampling process and can be widely applied into modern graph recommendation models. After a series of theoretical derivation, high-complexity terms in FGL can be effectively avoided, thus extensively accelerating the model learning process. Extensive experiments were conducted on the classic model matrix factorization (MF) and state-of-the-art graph neural network model LightGCN with four baseline datasets. Results shows that our optimization method is faster than BPR by several magnitudes, while acquires comparable even better accuracy performance, which indicates its potential of application in real-world in industrial tasks.

Key words recommendation system; graph neural networks; optimization framework; training efficiency

1 引言

这是一个推荐系统无处不在的时代。随着数字化的大力推进和大数据的迅猛发展, “信息过载”成为了当今不得不面对的问题。面对海量的数据, 用户迫切需要高效的推荐系统帮助他们快速获得感兴趣的信息。对推荐系统需求的涌现推动了学术界和工业界对其投入了大量的研究。

推荐系统中, 用户-物品交互图是研究的根本。这是天然存在于推荐系统中的二分图, 其中用户和物品作为图的结点, 交互行为作为图的边。推荐系统的基本任务就是从用户和物品的历史交互中学习用户的偏好和物品的特性。经典的基于交互图的推荐模型有矩阵分解(matrix factorization, MF)[1]等, 自此初步开始利用图来建模用户物品交互。近些年来。深度学习迎来了日新月异的发展, 研究者们将深度学习的方法应用于推荐系统中, 取得了令人瞩目的效果。在文献[2]中, 作者用多层感知机取代传统的内积来模拟用户和物品的交互; 文献[3]更是将注意力机制引入推荐系统。近来图卷积网络(graph convolution network, GCN)在图分类和结点辨识等任务上取得了重大突破[4-5], 吸引了许多学者的注意, 推荐系统中交互图的存在, 使得图卷积网络在推荐任务上大有作为。许多基于图卷积网络的推荐模型如 neural graph collaborative filtering(NGCF)[6], LightGCN[7]等, 应用图卷积操作在交互图上传递用户和物品特征, 大大提高了推荐的准确性。越来越多的研究表明, 相比于传统的推荐模型来说, 基于深度学习的推荐模型能取得更好表现。

推荐系统模型设计完成之后, 需要进一步优化以取得优异的表现, 此时需要损失函数来重建用户和物品的历史交互记录, 进而学习模型的参数。在推荐系统中, 广泛使用的是贝叶斯个性化排序(Bayesian personalized ranking, BPR)损失函数[8], 其思想是通过最大化已观测交互与未观测交互之间的评分差来优化模型, 现已在研究和应用中都取得了优异的效果。但其在数十年之前机器学习尚未流行的时代已经提出, 需要人工计算损失函数的梯度来学习模型参数。而当前业界更多是在能够自动微分的深度学习平台如 TensorFlow、Pytorch 上进行模型的自动优化, 并且一般采用批量梯度下降的方法, 从而充分利用 GPU 的并行计算实现优化操作, 这些是 BPR 设计之初没有考虑的。现有的大多数研究都集中于将深度学习应用在推荐系统的模型设计上, 然而随着模型设计的日渐复杂, 却几乎没有工作探索现有优化框架的不足, 更未曾提出新的解决办法。

本文的重点放在了模型训练的优化框架上, 尤其是 GPU 上基于图推荐模型的学习。我们分析了 BPR 在 GPU 上训练效率低下的原因。BPR 的基本流程是对于一个用户, 从其交互过的物品中采样一个作为正样本, 从其未交互过的物品中采样一个作为负样本, 由此形成一个正负样本对, 然后最大化正样本评分高于负样本评分的概率。为了简单验证 BPR 无法充分利用 GPU 的性能, 我们使用 BPR 训练矩阵分解模型 MF: 在 Amazon-book 数据集上, 设置相同的模型超参, 在 CPU(i9 9000kf)上进行一次迭代仅耗时 1.1 s, 而在 GPU(2080Ti)上进行一次迭代需要 55 s。可以明显看出, 模型在 GPU 上进行一次迭代花费了更多的时间, 其原因在于 BPR 计算无法充分利用 GPU 的并行计算。更准确的来说, 对于每一个正负样本对, BPR 都需要计算它们的评分差, 然后单独通过非线性激活函数比如 sigmoid, 这种对多个样本损失函数求和的计算方式, 无法转化成向量或者矩阵运算的形式。而 GPU 的加速性能本就来自大量的向量和矩阵运算, 缺乏这些运算使得 BPR 难以利用 GPU 来提升训练效率, 这也成为了制约推荐模型在大规模实际场景中应用的瓶颈, 第 3 节将对此进行更深入的实验分析。尽我们所知, 虽然以往的工作提及了 BPR 效率低下[9], 但本文首次深入分析了 BPR 在 GPU 上得到效率难以提升的原因。

在研究了BPR不能充分利用GPU并行计算性能之后, 本工作致力于为推荐系统寻找更高效的优化方法。在对平方误差损失函数进行深入分析后, 我们提出了一种快速且通用的学习方法 FGL(fast and generic learner), 其计算在推荐系统的假设之下包含两部分: 第一部分是只包含正样本的元素级别的计算, 计算复杂度较低; 第二部分包含所有的用户-物品对, 计算复杂度较高, 但通过线性代数的知识, 可以将其完全转化为矩阵运算的形式, 在理论上降低计算复杂度, 并能充分利用 GPU 对矩阵运算的加速性能。除了加速优化过程外, FGL 会用到所有的负样本, 因此相比于 BPR 节省了负采样时间。利用所有负样本还能够更充分地从负样本中提取有用信息, 因而在负样本比例较大, 即数据集稀疏的情况下, FGL 将取得更好的性能, 实验中也验证了这一假设。在实际场景中, 推荐系统的数据集往往是非常稀疏的, 因此 FGL 可以在工业界发挥巨大潜力。

本文挑选矩阵分解 MF[1]和 LightGCN[7]作为经典的图推荐模型和基于图卷积网络推荐模型的代表, 并将 FGL 应用其中。大量的实验结果表明在使用GPU进行训练时, 新的优化框架FGL相比于BPR而言能够带来数量级层面上的效率提升, 并且在较稠密数据集上准确性表现与 BPR 不相上下, 在较稀疏数据集上会比 BPR 取得更好的推荐结果。本文所用到的的数据集和代码公布在: https://github.com/YangZhengyi98/FGL。

本文的主要贡献如下:

(1) 我们首次深入分析了推荐系统中主流的优化框架 BPR 在利用 GPU 进行训练时效率低下的原因。

(2) 我们在推荐系统的框架下改进了平方误差损失函数, 并通过代数学知识将其转化为计算复杂度更低的矩阵运算的形式, 并提出我们的优化框架 FGL, 大大提高了 GPU 的利用率。

(3) 我们通过大量的实验验证了 FGL 的准确性和高效性, 并初步为其寻找了合适的应用场景。

2 推荐系统简介

我们关注基于协同过滤(collaborative filtering)[10]的推荐系统。协同过滤假设有过相似行为的用户会喜欢相似的物品, 因此可以从用户和物品的历史交互记录中捕捉用户的兴趣。我们用 $\mathcal{I}$ 表示所有物品的集合, $\mathcal{U}$ 表示所有用户的集合。$\mathcal{O}^+$ 表示所有观测到的用户-物品交互的集合(即正样本), 其中 $(u,i,y_{ui})\in\mathcal{O}^+$ 表示用户 u 对物品 i 的评价分数为 y_{ui}。一个推荐系统模型可以抽象表示为一个函数 $f:\mathcal{U}\times\mathcal{I}\to\mathbb{R}$, 即给定一个用户-物品对 (u,i), 推荐系统会输出一个分数 $\hat{y}_{ui}$, 表示模型预测的用户 u 对物品 i 的感兴趣程度, $\hat{y}_{ui}$ 越大表示系统推测用户 u 越有可能选择物品 i。推荐模型 f 通常用一系列参数进行建模, 一般来说我们将一个用户 u 对应到一个用户向量 $\boldsymbol{e}_u$, 将一个物品 i 对应到一个同维度的物品向量 $\boldsymbol{e}_i$, 并通过用户向量和物品向量来重建用户和物品间的历史交互记录。MF 模型中用户向量和物品向量通过随机初始化得到, LightGCN 模型中通过图卷积网络得到。

在深度学习尚未开始流行的时代, 推荐系统早期的研究包含了若干具有自己特色的观点: (1)数据方面。不同于显式反馈, 即用户对物品的喜好程度以评分的形式呈现, 更多的数据是隐式反馈[11]: 我们认为用户的不同行为都能体现用户的兴趣, 比如说点击、评论、购买、浏览等, 因此用户对物品的兴趣不再以高低不同的评分表示, 而是分为了感兴趣(正样本)和不感兴趣(负样本)两类。正样本从历史交互记录中很容易得到, 负样本在的获得在推荐系统中一般基于如下假设: 一个用户没有交互过的物品, 都认为用户对其不感兴趣。(2)损失函数方面。基于数据的正负样本, 两种类型的损失函数都被大量使用: 单点(point-wise)损失函数[11-12]通常遵循回归框架, 即最小化预测值与真实值之间的距离; 配对(pair-wise)损失函数[8,13]的观点是正样本的评分应该高于负样本, 因而要最大化它们的评分差。(3)优化方面。模型参数的学习通过对损失函数的优化来实现, 经典的做法是先人工计算出损失函数的梯度, 再用梯度下降法对其优化。上述观点都是在数十年前提出的, 当时机器学习和深度学习尚未兴起, 在 CPU 上进行实验还是主流。如今学术界和工业界的实验都通过 GPU 在深度学习平台如 TensorFlow 和 Pytorch 上进行, 因此直接将其搬移过来并不合适。

3 BPR 效率分析

3.1 BPR 公式

BPR(Bayesian personalized ranking)是推荐系统中基于隐式反馈的配对(pair-wise)损失函数的代表。其核心思想是对于一个用户的一个正样本, 随机取样一个负样本, 组成一个训练对, 然后最大化正样本与负样本预测评分的差值。在 GPU 常用的分批训练(mini-batch training)策略中, 如公式(1)所示:

$$L_{\text{BPR}}=\sum_{u\in\mathcal{B}}-\ln\sigma(\hat{y}_{ui}-\hat{y}_{uj}), \tag{1}$$

其中, $\mathcal{B}$ 表示一批训练的用户, 其中每个用户都从 $\mathcal{O}^+ = \{(u,i,y_{ui})\}$ 中采样一个感兴趣物品 i, 再从 $\mathcal{O}^- = \mathcal{U} \times \mathcal{I} \setminus \mathcal{O}^+$ 中采样一个不感兴趣的物品 j; $\sigma(\cdot)$ 表示 sigmoid 函数。遵循主流方法, 用户 u 对物品 i 的预测分数用对应用户向量和物品向量的内积表示, 即:

$$\hat{y}_{ui} = e_u^\top e_i. \tag{2}$$

通常会用 SGD[14]或者 Adam[15]作为优化器来进行梯度反向传播, 从而更新模型的参数。

3.2 原因分析

然而, 实验中发现在 GPU 上采用分批训练的策略时, 用 BPR 作为损失函数来训练模型会耗费大量的时间, 尤其是随着深度学习方法在推荐系统中得到广泛应用, 推荐模型逐渐复杂, 更使得训练模型的效率十分低下。我们认为这种低效性源于 BPR 的负采样策略和 BPR 损失函数的公式: (1)在一个训练批次中, 对于每一个用户, 都要对其采样一个正样本和负样本, 这种采样操作在 GPU 上不能被并行处理, 因而会花费很多时间; (2)在分批次训练中, 对于每个正负样本对, BPR 都要对其单独计算评分差, 然后通过非线性激活函数 sigmoid, 得到一个正负样本对的损失, 然后再将所有样本对的损失函数求和。对多个样本损失函数求和的计算方式使得 BPR 损失函数不能进行更好地向量化和矩阵化, 因此 GPU 难以通过直接对参数矩阵操作展现其卓越的并行计算能力, 所以 BPR 即使在 GPU 上也难以通过分批训练提升效率。

3.3 实验分析

为了验证对BPR效率低下的原因分析, 我们在图神经网络推荐模型 LightGCN 上进行了一系列实验。

首先要探讨负采样过程, 以及由非线性激活函数而导致的计算难以并行化对训练效率的影响。实验直接用 LightGCN 工作公开的代码, 和主流推荐模型一样, 其工作以 BPR 作为优化框架。我们采用 Yelp2018 和 Amazon Book(简写为 Book)两个推荐系统中常用的数据集, 数据集的介绍详见 5.1.1 节。我们将一次迭代中负采样的时间以及非线性激活函数消耗的时间展示在表 1 中, 实验中非线性激活函数消耗的时间通过计算移除 BPR 中非线性激活函数前后的时间差得到。

从表 1 中可以发现, 在 Yelp2018 数据集上, 一次迭代中负采样的时间大约占总时间的 1/5, 在 Amazon Book 上负采样时间约占 1/6, 可见负采样操作的确会在训练时消耗相当一部分时间。但与此同时, 移除了非线性激活函数之后, 训练的时间并没有显著下降。究其原因在于, 简单移除非线性激活函数并不能使 BPR 的计算得到向量化和矩阵化, 即移除非线性激活函数之后, BPR 仍需要对每个样本对的损失单独计算之后再求和, 计算还是元素级别的, 依然不能通过并行计算加速。因此为了提升训练效率, 必须在计算时引入更多向量化矩阵化的操作, 从而更好地利用 GPU 的并行计算性能。

表 1 一次迭代 BPR 中负采样和非线性激活函数操作的时间比较

Table 1 Time comparison of negative sampling and nonlinear activation function in BPR in one epoch

	总时间(s)	负采样时间(s)	非线性激活函数时间(s)
Yelp2018	222	56	17
Book	831	135	39

为了验证上述想法, 我们人为地在计算中引入更多的向量化矩阵化等能够并行处理的操作, 我们对公式(1)加以修改, 并将其应用于 LightGCN。引入并行操作的方法是: 给定一个训练批次中的一个用户, 采样 T 个和他有过交互的物品和 T 个和他没有交互的物品, 并最大化它们评分和之间的差距, 修改后的公式如式(3)所示:

$$\begin{aligned}\tilde{L}_{\text{BPR}} &= \sum_{u\in\mathcal{B}} -\ln\sigma\Big(\sum_{t=1}^{T}\hat{y}_{ui_t} - \sum_{t=1}^{T}\hat{y}_{uj_t}\Big)\\ &= \sum_{u\in\mathcal{B}} -\ln\sigma\Big(\sum_{t=1}^{T}e_u^\top e_{i_t} - \sum_{t=1}^{T}e_u^\top e_{j_t}\Big)\\ &= \sum_{u\in\mathcal{B}} -\ln\sigma\Big(e_u^\top\Big(\sum_{t=1}^{T}e_{i_t} - \sum_{t=1}^{T}e_{j_t}\Big)\Big),\end{aligned} \tag{3}$$

其中 $\{i_t\}_{t=1}^T$ 和 $\{j_t\}_{t=1}^T$ 分别表示用户 u 有过交互的物品和没有交互的物品。显然, 原始的 BPR 损失就是修改后的 BPR 损失在 $T=1$ 时的特例。从修改后的公式中可以看出, 通过将非线性激活函数 sigmoid 中的一个用户匹配多个正样本和多个负样本, 损失函数的计算将在通过非线性激活函数之前具有更高程度的向量化, 而向量化的计算能够更加充分地利用 GPU 的并行计算能力。

仍在 LightGCN 工作的代码上进行实验。为了对比的公平性, 除对损失函数进行了如公式(3)的修改之外, 不对其他实验设置做任何更改。修改前后准确性和效率的比较分别展示在表 2 和表 3 中。

从表 2 中可以看出, 随着 T 的增加, 两个数据集上 Recall@20 和 NDCG@20 都在降低。这是在预料之中的, 因为 BPR 经过了严格的数学推导论证, 直接将多个物品向量加起来再经过非线性激活函数没

表 2 修改前后 BPR 损失函数推荐结果的比较

Table 2 Performance comparison between the original BPR an revised BPR

	Yelp2018		Book	
T	R@20	N@20	R@20	N@20
1	0.0644	0.0528	0.0410	0.0318
2	0.0581	0.0475	0.0347	0.0272
4	0.0467	0.0385	0.0256	0.0206
8	0.0380	0.0313	0.0206	0.0163
16	0.0345	0.0281	0.0163	0.0130
32	0.0328	0.0271	0.0135	0.0107

(注: R@20 为 Recall@20, N@20 为 NDCG@20; *T*=1 即表示原始的 BPR)

表 3 修改前后 BPR 损失函数效率的比较

Table 3 Efficiency comparison between the original BPR an revised BPRs

	Yelp2018			Book		
T	TC(s)	NE	TT(h)	TC(s)	NE	TT(h)
1	222	520	32.1	831	340	78.5
2	119	360	11.9	411	680	77.6
4	61	640	10.8	213	640	37.9
8	38	700	7.4	108	1240	37.2
16	27	820	6.2	67	1000	18.6
32	22	800	4.9	50	1180	16.4

(注: TC 为每次迭代时间, NE 为总迭代数, TT 为总训练时间; s 代表秒, h 代表小时; *T*=1 即表示原始的 BPR)

有理论保证, 不可避免地会对推荐的准确性表现产生负面影响。然而, 如表 3 所示, 增加 *T* 在两个数据集上都能够大大缩短训练时间, 比如在 Amazon-book 上, *T*=1(即原始的 BPR)时训练需要 78.5 h, 而 *T*=8 时下降到了 37.2 h。这说明了原始的 BPR 中, 每次将一个正负样本对的评分差单独通过非线性激活函数在 GPU 上是很低效的。随着 *T* 的增加, 越来越多正负样本对评分和的差值一起通过非线性激活函数, 增加了每次非线性操作之前计算的向量化程度, 能够更好地利用 GPU 的并行计算能力, 因此能够大大减少训练时间。

通过对实验现象的观察和上述的讨论可知: 原始 BPR 的确存在效率低下的问题。修改后的 BPR 由于在计算中增加了更多的向量化运算, 能更高效地利用 GPU 的并行计算性能, 提高训练效率, 但鲁莽地在计算中增加向量化的操作并没有理论依据, 从而带来推荐准确性的损失。因此我们需要寻找其他策略, 来解决损失函数的计算中向量化矩阵化程度较低这一问题。

4 FGL

第三节我们对 BPR 损失函数进行了深入研究, 表明原始的BPR对GPU利用率低的问题存在于其公式之中, 修改后的 BPR 虽然能够更加充分地利用 GPU, 但却牺牲了推荐准确性。受以上发现的启发, 本文致力于为推荐系统探寻新的优化框架, 使得在提高模型训练速度的同时, 保持推荐结果的准确性。

在不断地探寻中我们发现, 可以基于加权平方误差损失函数, 为推荐系统设计新的优化框架, 并基于应用全部负样本的策略, 经过严格的数学推导, 将其转化为计算复杂度更低的矩阵运算的形式, 从而提高 GPU 的利用率。我们将其命名为 FGL(fast and generic learner)。

4.1 从平方误差到 FGL

加权平方误差是在平方误差的基础上, 根据训练样本的重要性不同, 为其分配不同的权重, 是机器学习领域中常用的优化函数, 如公式(4)所示:

$$L_{LMS} = \sum_{(u,i,y_{ui})\in\mathcal{O}} c_{ui}(y_{ui} - \hat{y}_{ui})^2, \tag{4}$$

其中, $\mathcal{O}$ 表示所有训练样本的集合,既包含正样本又包含负样本。c_{ui} 表示用户 u 和物品 i 对应的训练样本的权重, 其大小与对应样本的重要性有关。

将平方误差与推荐系统结合的思想最早可以追溯到文献[16], 作者简单将最小均方误差应用于推荐系统任务, 但由于当时 GPU 计算尚未流行, 作者并没有做进一步分析, 更不用说推导出更适合 GPU 运算的矩阵形式。最近的文献[9]采用了和文献[16]同样的思路, 改进的地方在于使用 MLP 取代内积来计算用户对物品的预测评分, 对于损失函数的具体分析只进行了元素层面的推导, 也未进一步结合 GPU 的特性进一步推导出矩阵层面的算法。

我们在文献[16]的启发下, 将平方误差与推荐系统结合, 以充分利用 GPU 计算能力为目标, 最终得到了我们的方法 FGL。

与 BPR 相似, 考虑基于隐式反馈的推荐任务, 即用户对物品的感兴趣程度是二值的, 只有喜欢和不喜欢之分, 那么有:

$$y_{ui} = \begin{cases} 1, & \text{用户}u\text{与物品}i\text{有交互} \\ 0, & \text{用户}u\text{与物品}i\text{无交互} \end{cases}. \tag{5}$$

对于权重来说, 不同的交互可以体现用户不同的偏好, 比如购买比浏览更能体现用户的喜好, 因此不同训练样本的权重理应有所不同。同样基于隐式反馈的假设, 认为权重也是二值的, 将正样本的权重设置为 α, 将负样本的权重设置为 1, 即:

$$c_{ui}=\begin{cases}\alpha, & \text{用户}u\text{与物品}i\text{有交互}\\ 1, & \text{用户}u\text{与物品}i\text{无交互}\end{cases}. \tag{6}$$

α 是我们引入的超参数, 将在实验部分对其加以研究。关于 α 的取值, 正样本中的用户和物品是发生过交互的, 而负样本中用户和物品没有过交互, 但没有交互也可能是因为物品没有曝光给用户, 即负样本中噪声较大, 因此正样本的权重应该比负样本的权重要大, α 的取值应大于 1。

公式(4)中, 训练样本既包括正样本又包括负样本, 不同于 BPR 中正负样本的比例为 1∶1, 这里使用所有的负样本, 如此不仅可以节省负采样的时间, 也有利于接下来的推导。基于以上讨论, 将推荐系统知识与应用在公式(4)中, 可以进行如式(7)中的推导:

$$\begin{aligned}
L^*_{LMS}&=\sum_{u\in\mathcal{U}}\sum_{i\in\mathcal{I}}c_{ui}(y_{ui}-\hat{y}_{ui})^2 \quad (7.a)\\
&=\sum_{u\in\mathcal{U}}\sum_{i\in\mathcal{I}}c_{ui}(y_{ui}^2-2y_{ui}\hat{y}_{ui}+\hat{y}_{ui}^2) \quad (7.b)\\
&=C-2\sum_{u\in\mathcal{U}}\sum_{i\in\mathcal{I}}c_{ui}y_{ui}\hat{y}_{ui}+\sum_{u\in\mathcal{U}}\sum_{i\in\mathcal{I}}c_{ui}\hat{y}_{ui}^2 \quad (7.c)\\
&=C-2\sum_{u\in\mathcal{U}}\sum_{i\in\mathcal{I}_u^+}\alpha y_{ui}\hat{y}_{ui}+\sum_{u\in\mathcal{U}}\sum_{i\in\mathcal{I}_u^+}\alpha\hat{y}_{ui}^2\\
&\quad+\sum_{u\in\mathcal{U}}\sum_{i\in\mathcal{I}_u^-}\hat{y}_{ui}^2 \quad (7.d)\\
&=C-2\sum_{u\in\mathcal{U}}\sum_{i\in\mathcal{I}_u^+}\alpha y_{ui}\hat{y}_{ui}+\sum_{u\in\mathcal{U}}\sum_{i\in\mathcal{I}_u^+}\alpha\hat{y}_{ui}^2\\
&\quad+\sum_{u\in\mathcal{U}}\sum_{i\in\mathcal{I}_u}\hat{y}_{ui}^2-\sum_{u\in\mathcal{U}}\sum_{i\in\mathcal{I}_u^+}\hat{y}_{ui}^2 \quad (7.e)\\
&=C+\sum_{u\in\mathcal{U}}\sum_{i\in\mathcal{I}_u^+}[(\alpha-1)\hat{y}_{ui}^2-2\alpha\hat{y}_{ui}]\\
&\quad+\sum_{u\in\mathcal{U}}\sum_{i\in\mathcal{I}_u}\hat{y}_{ui}^2 \quad (7.f),
\end{aligned} \tag{7}$$

其中, C 为常数, $\mathcal{U}$ 为所有的用户集合, $\mathcal{I}$ 为所有的物品集合, $\mathcal{I}_u^+$ 是由用户 u 所有交互过的物品构成的集合, $\mathcal{I}_u^-$ 是由用户 u 所有未交互过的物品构成的集合。式(7.a)是加权平方误差损失函数原始公式; 式(7.b)做了简单的展开; 式(7.c)中的常数是因为对于给定的数据集来说, 哪些用户对哪些物品产生交互行为是确定的, 因此上式中的第一项可以用常数表示; 式(7.d)中的第二项是因为在式(7.c)的第二项中, 对于负样本来说 $y_{ui}=0$, 因而可以只保留正样本, 第三项和第四项是将式(7.c)中的正负样本分开; 式(7.e)是将式(7.d)中所有负样本预测评分的平方和, 分解为所有样本的平方和减去所有正样本的平方和的形式; 式(7.f)是对具有相同求和下标的项进行合并。将常数项省略, 得到新的损失函数公式如式(8):

$$L^*_{LMS}=\sum_{u\in\mathcal{U}}\sum_{i\in\mathcal{I}_u^+}[(\alpha-1)\hat{y}_{ui}^2-2\alpha\hat{y}_{ui}]+\sum_{u\in\mathcal{U}}\sum_{i\in\mathcal{I}_u}\hat{y}_{ui}^2. \tag{8}$$

从求和范围可以看出, 式(8)的第一项是关于所有的正样本的求和项, 推荐系统数据集稀疏的特性使得第一项的计算复杂度很低; 第二项是关于所有的用户-物品对的求和项, 直接计算的话复杂度较高。之前的工作如文献[9, 16]仅仅止步于此, 仍然采用元素层面的运算, 不能充分利用 GPU 的性能, 因此我们接下来将对其进行进一步的探究。

为了推导式(8)的矩阵计算形式, 我们先给出一些定义: 定义矩阵 $\boldsymbol{P}\in\mathbb{R}^{|\mathcal{U}|\times d}$ 为所有用户向量构成的矩阵, $|\mathcal{U}|$ 为用户的数目, d 为用户向量的维度, 其中 $\boldsymbol{P}$ 的第 k 行表示第 k 个用户的用户向量 $\boldsymbol{e}_{u_k}$。同理定义 $\boldsymbol{Q}\in\mathbb{R}^{|\mathcal{I}|\times d}$ 为所有物品向量构成的矩阵, 每一行对应一个物品向量, 接下来对式(8)中第二项进行如下推导:

$$\begin{aligned}
\sum_{u\in\mathcal{U}}\sum_{i\in\mathcal{I}}\hat{y}_{ui}^2&=\sum_{u\in\mathcal{U}}\sum_{i\in\mathcal{I}}(\boldsymbol{e}_u^\top\boldsymbol{e}_i)^2\\
&=\sum_{u\in\mathcal{U}}\sum_{i\in\mathcal{I}}\boldsymbol{e}_u^\top\boldsymbol{e}_i\boldsymbol{e}_i^\top\boldsymbol{e}_u\\
&=\sum_{u\in\mathcal{U}}\boldsymbol{e}_u^\top(\sum_{i\in\mathcal{I}}\boldsymbol{e}_i\boldsymbol{e}_i^\top)\boldsymbol{e}_u\\
&=\sum_{u\in\mathcal{U}}\boldsymbol{e}_u^\top(\boldsymbol{Q}^\top\boldsymbol{Q})\boldsymbol{e}_u\\
&=\mathrm{Tr}(\boldsymbol{P}\boldsymbol{Q}^\top\boldsymbol{Q}\boldsymbol{P}^\top).
\end{aligned} \tag{9}$$

上式推导中用到了式(2)中给出的预测评分的定义, 以及矩阵迹的知识, 不再做详细解释。另外, 线性代数中矩阵的迹有如下性质: 对于两个维数适当的矩阵 $\boldsymbol{A}$ 和 $\boldsymbol{B}$ 来说:

$$\mathrm{Tr}(\boldsymbol{A}\boldsymbol{B})=\mathrm{Tr}(\boldsymbol{B}\boldsymbol{A}). \tag{10}$$

应用这一性质, 我们对式(9)做进一步变换:

$$\begin{aligned}
\mathrm{Tr}(\boldsymbol{P}\boldsymbol{Q}^\top\boldsymbol{Q}\boldsymbol{P}^\top)&=\mathrm{Tr}(\boldsymbol{P}^\top\boldsymbol{P}\boldsymbol{Q}^\top\boldsymbol{Q})\\
&=\mathrm{Tr}((\boldsymbol{P}^\top\boldsymbol{P})(\boldsymbol{Q}^\top\boldsymbol{Q})).
\end{aligned} \tag{11}$$

最终将 FGL 的表达式写为:

$$L^*_{FGL}=\sum_{u\in\mathcal{U}}\sum_{i\in\mathcal{I}_u^+}[(\alpha-1)\hat{y}_{ui}^2-2\alpha\hat{y}_{ui}]+\mathrm{Tr}((\boldsymbol{P}^\top\boldsymbol{P})(\boldsymbol{Q}^\top\boldsymbol{Q})). \tag{12}$$

容易看出, 损失函数的第一项包含了全部的正样本, 第二项是用户向量矩阵和物品向量矩阵的一系列线性操作。

式(9)~式(12)是我们工作的核心, 也是区别于文献[9, 16]的独到之处, 我们利用线性代数知识将复杂度很高的项转化为一系列矩阵的线性运算, 并利用矩阵迹的性质交换矩阵相乘的顺序。我们知道矩阵相乘的时间复杂度与相乘顺序密切相关, 而对矩阵的线性运算正是 GPU 所擅长的, 因此 L^*_{FGL} 很容易部署在当今常用的深度学习平台如 TensorFlow 上, 并

适合在 GPU 上运行。

4.2 计算复杂度分析

我们将加权平方误差损失函数在推荐系统的框架下, 运用严格的数学推导, 最终得到了 FGL 表达式如式(12)所示, 下面将对其进行计算复杂度分析。

直接分析使用所有负样本的情况。原始的平方误差如公式(4)所示, 其考虑了所有的用户-物品对, 因为用户, 物品向量都是 d 维的, 所以其计算复杂度为 $O(|\mathcal{U}||\mathcal{I}|d)$ 。从公式(4)到公式(7)仅仅是将推荐系统的一些假设应用于最小均方误差损失函数, 并省去了一些常数项, 理论上复杂度并未降低。真正降低复杂度是在公式(9)将矩阵的迹引入之后, 我们在公式(11)中应用了迹的性质, 改变了矩阵相乘的顺序。如果按公式(9)中的顺序进行相乘, 计算复杂度为 $O(|\mathcal{U}||\mathcal{I}|d+|\mathcal{U}||\mathcal{I}|d+|\mathcal{U}||\mathcal{I}|d)$, 而按公式(11)中的顺序的话, 计算复杂度将变为 $O(|\mathcal{U}|d^2+|\mathcal{I}|d^2+d^3)$ 。在实际的推荐系统数据集中, 用户和物品的数量相当, 能达到几万甚至几十万, 而推荐模型中用户和物品向量的维度一般不会超过 256, 因此 $d \ll |\mathcal{U}|,|\mathcal{I}|$ 。所以式(9)的计算复杂度可以等效为 $O(|\mathcal{U}||\mathcal{I}|d)$, 将远远大于式(11)的计算复杂度 $O(|\mathcal{U}|d^2)$ 。以上都是对损失函数的第二项进行讨论的, 现在来讨论第一项, 推导前后(式(8)和式(12))第一项的复杂度均为 $O(|\mathcal{O}^+|d)$, 从推荐系统数据稀疏的特性可以知道 $|\mathcal{O}^+| \ll |\mathcal{U}||\mathcal{I}|$, 因而第一项的计算复杂度相比于第二项可以省略。由以上讨论可知, 经过严格的数学推导, 理论上 FGL 的计算复杂度将大大降低。

另外, 由于 GPU 能够对矩阵计算进行天然的并行加速, 而我们又通过严格的数学推导将加权平方误差损失函数中计算复杂度最高的项表示成了矩阵线性运算的形式, 因此 FGL 能更充分地利用 GPU 的性能。

4.3 与 BPR 比较

对 BPR 损失函数进行深入分析得出了 BPR 效率低下的原因来自两个方面: 一是对每个正样本负采样得到一个负样本, 负采样过程会浪费 GPU 资源; 二是要对每个正负样本对的评分差单独通过非线性激活函数, 不利于 GPU 进行并行加速。

而 FGL 能够完美地解决上面两个问题: (1)FGL 会使用所有的正负样本, 不用进行负采样, 从而不会因为负采样降低训练效率; (2)FGL 中复杂度最高的项是矩阵线性运算的形式, 能高效利用 GPU 快速矩阵运算的特性。

此外, 从理论上, 相比于 BPR 损失而言, FGL 在稀疏的数据上表现将会更好。原因在于 BPR 中正负样本的比例是 1∶1, 而 FGL 会用到所有的负样本, 即 FGL 利用了更多的负样本, 从而理论上能够从负样本中学到更多有用的信息。数据集越稀疏, 负样本的比例就越大, 因此 FGL 在稀疏数据集上将会有更好的表现, 这也是经过简单分析后得出的 FGL 应用场景。实际中推荐系统的数据集往往非常稀疏, 因而预测 FGL 在实际情况中会有更好的表现。我们将 FGL 应用场景的深入研究留给未来工作。

5 实验流程与结果分析

为了评估 FGL 的表现和效率, 并验证 FGL 的适用场景, 我们在 4 个来自实际的数据集上进行了大量的实验。

5.1 实验设置

5.1.1 数据集

实验中使用了 4 个来自真实世界的数据集: Yelp2018, Amazon Book(简写为 Book), Amazon Office Products(简写为 Office), Amazon Cellphones and Accessories(简写为 Cellphones)。其中前两个选自自 LightGCN 工作, 用户-物品交互较为稠密。为了体现我们的方法在大规模场景下的优势, 并比较数据集稀疏度对方法的影响, 我们另选取 Amazon 数据集的 2 个子集, 相较于前 2 个数据拥有更大的用户物品基数, 但是用户物品交互较为稀疏。我们在表 4 中展示了数据集的统计特性:

表 4 数据集统计特性

Table 4 Statistics of the datasets

数据集	用户数	物品数	交互数	稠密度
Yelp2018	31668	38048	1561406	0.00130
Book	52643	91599	2984108	0.00062
Office	138728	136257	1111148	0.00006
Cellphones	241475	270452	1793308	0.00003

下面对数据集进行简要介绍:

(1) Yelp2018: 数据来自 2018 版本的 Yelp Challenge, 其中餐馆、酒吧等被当作物品。

(2) Book, Office, Cellphones: 数据来自著名推荐数据集 Amazon review, 3 个数据的物品分别是书籍, 办公用品, 和手机等电子产品。

对于每个数据集, 我们从每个用户历史交互物品中随机挑选 80%作为训练集, 其余 20%作为测试集。

5.1.2 模型介绍

我们在实验中使用了推荐系统中经典的图模型 MF, 和目前最先进的基于图卷积网络的推荐模型 LightGCN:

(1) MF[1]模型中, 用于计算评分的用户向量和物品向量通过随机初始化得到, 并在训练时对它们进行优化。

(2) LightGCN[7]模型中, 先随机初始化一组原始的用户向量和物品向量, 用于计算评分的用户, 物品向量通过原始向量经过用户-物品交互二分图进行图卷积得到, 图卷积的层数与 LightGCN 的层数相对应。

5.1.3 评价指标

模型训练之后, 对于每个用户, 我们对其所有未在训练集中交互的物品计算预测评分, 然后按评分由高到低对这些物品排序, 挑选出排在前 N 位的物品, 作为预测的用户感兴趣的物品, 即推荐系统中常用的 Top-N 评估[8], 具体评价指标采用 Recall@N 和 NDCG@N[6], 其中 Recall@N 只考虑了用户在测试集中感兴趣的物品有没有在前 N 位出现, 而不考虑出现的位置先后; 而 NDCG@N 不仅考虑了预测前 N 位中真正感兴趣的物品的比例, 还要考虑相对位置因素, 认为把感兴趣的物品排得靠前的模型要更好。实验中取 N 为 20 和 100。计算出每个用户的指标后, 对所有的用户取平均得到总指标。

5.1.4 超参数设置

所有的实验都在 TensorFlow 平台实现。在数据集 Yelp2018 和 Book 上, 用户和物品向量的维度设置为 64; 其他两个数据集 Office 和 Cellphones 规模较大, 用户物品数量都多达几十万, 因而将物品向量的维度设置为 16, 以避免内存不足的问题。对于 BPR 来说, 采用分批训练的策略, 每批的大小(batch size)为 1024。两个方法优化器均使用 Adam。学习率等超参数采用网格搜索的策略选择最优的, 学习率从 $\{0.001, 0.01\}$ 中选择效果最好的; 均采用 L_2 正则化, 正则化系数从 $\{e^{-5}, e^{-4}, e^{-3}, e^{-2}, e^{-1}\}$ 中选择效果最好的。我们在 FGL 中引入了一个新的超参数 α 来表征正样本的权重, 经过讨论 α 取值应该大于 1, 因此在实验中搜索 α 的范围为: $\{10, 20, \cdots, 100\}$。对于 LightGCN 模型, 不失一般性, 下面展示了 2 层和 3 层的结果。

5.2 训练效率比较

本节将进行 FGL 与 BPR 损失函数的效率比较, 为了尽可能保证比较的公平性, 所有的实验都在相同的 GPU(GTX 1080Ti)上进行。我们把在四个不同的数据集上, 采用不同的图推荐模型的比较结果展示在表 5 中。

表 5 BPR 与 FGL 的效率比较

Table 5 Efficiency comparison between BPR and FGL

数据集		Yelp2018			Book			Office			Cellphones		
模型	损失函数	TC	NE	TT	TC	NE	TT	TC	NE	TT	TC	NE	TT
MF	BPR	39s	460	5.0h	118s	600	19.7h	46s	1910	24.4h	164s	1770	80.1h
	FGL	0.8s	900	12.0m	1.5s	100	2.5m	0.6s	3600	36.0m	1.0s	4500	1.25 m
LightGCN 2 layers	BPR	157s	1300	56.7h	562s	460	71.8h	118s	1540	50.5h	3322s	1460	134.6h
	FGL	0.8s	1800	24m	1.5s	500	12.5m	0.6s	1900	19m	1.0s	2500	41.7m
LightGCN 3 layers	BPR	222s	520	32.1h	831s	340	78.5h	140s	1240	48.2h	415s	940	108.4h
	FGL	0.8s	1250	16.7m	1.5s	870	21.8m	0.6s	1400	14.0m	1.0s	1900	31.7m

(注: TC 为每次迭代时间, NE 为总迭代数, TT 为总训练时间; s 代表秒, m 代表分钟, h 代表小时)

从表 5 中有一些重要的发现。首先, 对于每次迭代的训练时间来说, 从 MF 模型到 2 层 LightGCN 再到 3 层 LightGCN, 随着模型复杂度的增加, FGL 每次迭代的时间几乎不变, 而 BPR 每次迭代的时间大幅增加。比如在数据集 Yelp2018 上, 采用 BPR 损失函数, LightGCN 的层数从 2 层增加到 3 层, 每次迭代的时间从 157s 增加到了 222s; 而采用 FGL 每次迭代的时间都在 0.8s。两种优化框架不同的计算方式很好地解释这一现象。FGL 的计算是一系列矩阵的线性运算, LightGCN 中也恰好去掉了图卷积网络中的非线性激活函数, 因此即使随着层数的增加, LightGCN 的计算也是线性的, 所以即使层数增加了, 依然能够充分利用 GPU 的性能。但是非线性计算在 BPR 损失中占据了很大一部分, 随着 LightGCN 层数的增加, 对 GPU 利用率低下的问题会越来越明显, 因此 BPR 每次迭代的时间会随着层数的增加大幅增加。

在总训练时间上, FGL 损失相比于 BPR 损失来

说可以将训练速度提升几个数量级。虽然 FGL 会进行更多次数的迭代才能收敛, 但每个迭代耗时极少, 因此总的训练时长非常短。比如在数据集 Book 上, 模型用三层的 LightGCN 时, BPR 需要 78.5h 才能收敛, 而 FGL 只需要 21.8min, 即 FGL 将训练时间从数天缩短到了不到 0.5h。因为两种方法只有优化框架上的差异, 因此我们可以得出结论: FGL 相对于 BPR 损失来说的确能带来训练速度的极大提升。

5.3 准确性比较

上一节比较了 FGL 和 BPR 的训练效率, 得出 FGL 在效率上具有巨大的优势。本节将进一步比较两种方法的推荐准确性, 以说明 FGL 效率的提升并没有以牺牲推荐准确性表现为代价。准确性的表现总结在表 6 中。

表 6 BPR 与 FGL 的推荐准确性比较

Table 6 Recommendation accuracy comparison between BPR and FGL

数据集	稀疏度	模型	MF		LightGCN 2 layers		LightGCN 3layers	
		损失函数	BPR	FGL	BPR	FGL	BPR	FGL
Yelp2018	0.00130	R@20	0.0528	**0.0631(+19.5%)**	0.0622	**0.0644(+3.5%)**	0.0639	0.0636(–0.4%)
		R@100	0.1675	**0.1884(+12.4%)**	0.1874	**0.1934(+3.2%)**	0.01918	**0.1922(+0.2%)**
		N@20	0.0423	**0.0520(+22.9%)**	0.0504	**0.0528(+4.7%)**	0.0525	0.0522(–0.5%)
		N@100	0.0821	**0.0953(+16.0%)**	0.0932	**0.0976(+4.7%)**	0.0964	0.0963(–0.1%)
Book	0.00062	R@20	0.0325	**0.0363(+11.6%)**	0.0411	0.0381(–7.2%)	0.0410	0.0379(–7.5%)
		R@100	0.1051	**0.1088(+3.5%)**	0.1243	0.1160(–7.1%)	0.1233	0.1173(–4.8%)
		N@20	0.0250	**0.0288(+15.2%)**	0.0315	0.0299(–5.0%)	0.0318	0.0294(–7.5%)
		N@100	0.0499	**0.0538(+7.8%)**	0.0600	0.0567(–5.5%)	0.0608	0.0569(–6.4%)
Office	0.00006	R@20	0.0645	**0.1135(+75.9%)**	0.0841	**0.1163(+38.2%)**	0.0822	**0.1164(+41.6%)**
		R@100	0.1472	**0.1871(+27.1%)**	0.1704	**0.1905(+11.7%)**	0.1665	**0.1927(+15.7%)**
		N@20	0.0291	**0.0701(+140.8%)**	0.0424	**0.0712(+67.9%)**	0.0413	**0.0730(+76.7%)**
		N@100	0.0446	**0.0841(+88.5%)**	0.0686	**0.0856(+24.7%)**	0.0663	**0.0868(+30.9%)**
Cellphones	0.00003	R@20	0.0364	**0.0605(+66.2%)**	0.0520	**0.0614(+18.0%)**	0.0520	**0.0611(+17.5%)**
		R@100	0.1003	**0.1272(+26.8%)**	0.1239	**0.1282(+3.4%)**	0.1235	**0.1273(+3.0%)**
		N@20	0.0148	**0.0296(+100.0%)**	0.0232	**0.0307(+32.3%)**	0.0234	**0.0308(+31.6%)**
		N@100	0.0267	**0.0420(+57.3%)**	0.0364	**0.0433(+18.9%)**	0.0362	**0.0447(+23.4%)**

(注: R@20 为 Recall@20, R@100 为 Recall@100, N@20 为 NDCG@20, N@100 为 NDCG@100)

从表 6 中可以看出 FGL 的表现在 Yelp2018 和 Book 数据集上取得了和 BPR 不相上下的结果, 在 Office 和 Cellphones 数据集上, FGL 推荐结果明显好于 BPR。于是可以认为: FGL 在加快训练速度的同时, 并不会损失推荐结果的准确性。

虽然能看出 FGL 推荐准确性不逊色于 BPR, 但在不同的数据集上, 性能出现了明显的差异, 这促使我们进步分析其原因, 以给出 FGL 的适用场景。经过分析之后, 我们将其归因于数据集之间不同统计特性带来的差异。之前提到过, Yelp2018 和 Book 数据集直接来自于 LightGCN 的工作, 我们了解到作者在处理这两个数据集的时候, 只保留了原始数据中拥有 10 个以上交互的用户和物品, 从数据集的统计特性中我们也可以看出, 这两个数据集更为稠密。而本文为了得到规模更大的数据集, 在处理 Office 和 Cellphones 数据集时, 保留了拥有 5 个以上历史交互的用户, 物品不做过滤, 因此后两个数据集更为稀疏, 并保留了物品长尾分布的特性。而我们知道实际场景中推荐数据集往往非常稀疏并且长尾效应明显, 根据之前的分析, FGL 能够利用全部负样本信息, 因此将在这种情况下取得更好的效果。

5.4 超参数研究

我们在 FGL 中引入了超参数 α, 用来表示正样本的权重, 本节将研究 α 对于推荐结果的影响。不失一般性, 将使用 2 层 LightGCN 模型, 在 Yelp2018 和 Cellphones 数据集的表现绘制在图 1 中。

从图 1 中可以看出, 不同的 α 会对推荐结果产生很大的影响, 这也说明了在 FGL 中, 平衡正负样本对得到较好的推荐结果来说至关重要。此外, 在不同的数据集上, 达到最好表现的 α 也不相同, 对于 Yelp2018 数据集, 在 α 取 30 时达到最好表现; 而对于 Cellphones 数据集, 最好的 α 是 90。因此在实验或应用中, 使用 FGL 时, 应该认真调整 α 以得到最好的推荐效果。

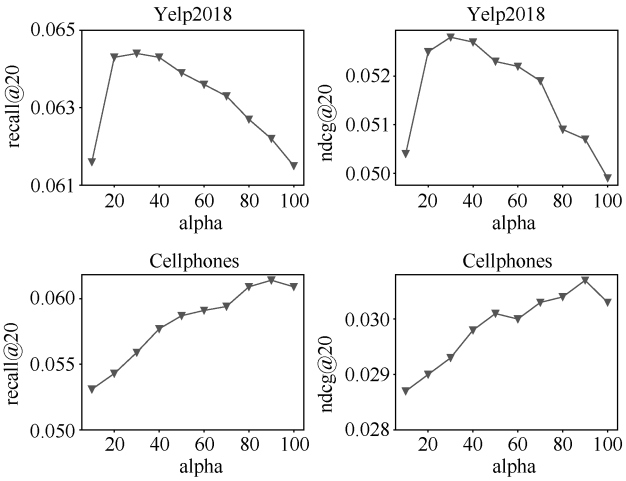

图 1　2 层 LightGCN 的推荐结果在 Yelp2018 和 Cellphones 上随 α 变化的曲线

Figure 1　Performance of 2-layer LightGCN w.r.t. different α on Yelp2018 and Cellphones

6　结论与未来工作

在本文中, 我们指出随着深度学习, 尤其是图卷积网络在推荐系统中的应用, 以及在 GPU 上进行模型训练的流行, 用 BPR 损失函数进行模型参数优化十分低效, 原因在于 BPR 对每个样本都要进行非线性变换, 缺乏矩阵化和向量化的计算, 浪费了 GPU 的并行计算性能。为了验证这一想法, 通过将多个样本的评分求和之后, 再通过非线性激活函数的方法, 我们人工地在 BPR 中加入向量化运算, 这种做法的确会使训练速度得到提升, 但是不出意外的损失了推荐的准确性。为了克服这一问题, 必须寻找 BPR 以外的优化框架。我们发现通过在推荐系统中引入加权平方误差损失函数, 加以严格的数学推导, 能够将其转化为一系列矩阵线性运算的形式, 并且可以利用线性代数的知识改变矩阵运算的顺序, 进一步降低其计算复杂度。据此我们提出了 FGL: 一种快速且普适的学习框架。除在计算方面更为高效外, FGL 还节省了 BPR 中负采样的时间。我们在四个真实数据集上进行了大量的实验, 结果表明在保持推荐准确性的情况下, FGL 能够大大提高训练效率。

未来我们将对这个工作的探索分为两个部分: 一是寻找更合适的应用场景, 我们在简单分析后得出 FGL 在数据集稀疏, 并且具有长尾特性时表现很好, 我们将在这方面进行更多研究, 以给出更具有指导意义的结论; 二是 FGL 中引入了超参数 α, 现在采用网格搜索的方法寻找最优值, 我们将研究能否采用更快捷的策略, 如自适应方法等来选择 α。

参考文献

[1] Koren Y, Bell R, Volinsky C. Matrix Factorization Techniques for Recommender Systems[J]. *Computer*, 2009, 42(8): 30-37.

[2] He X N, Liao L Z, Zhang H W. Neural Collaborative Filtering[EB/OL]. 2017: arXiv: 1708.05031[cs.IR]. https://arxiv.org/abs/1708.05031.

[3] He X N, He Z K, Song J K, et al. NAIS: Neural Attentive Item Similarity Model for Recommendation[J]. *IEEE Transactions on Knowledge and Data Engineering*, 2018, 30(12): 2354-2366.

[4] William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive Representation Learning on Large Graphs[C]. *NIPS*, 2017: 1025–1035.

[5] Kipf T N, Welling M. Semi-Supervised Classification with Graph Convolutional Networks[EB/OL]. 2016: arXiv: 1609.02907[cs.LG]. https://arxiv.org/abs/1609.02907

[6] Liu Z W, Meng L, Jiang F, et al. Deoscillated Graph Collaborative Filtering[EB/OL]. 2020: arXiv: 2011.02100[cs.IR]. https://arxiv.org/abs/2011.02100

[7] He X N, Deng K, Wang X, et al. LightGCN: Simplifying and Powering Graph Convolution Network for Recommendation[C]. *The 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2020: 639–648.

[8] Rendle S, Freudenthaler C, Gantner Z, et al. BPR: Bayesian Personalized Ranking from Implicit Feedback[C]. *UAI*, 2009: 452– 461.

[9] Chen C, Zhang M, Zhang Y F, et al. Efficient Neural Matrix Factorization without Sampling for Recommendation[J]. *ACM Transactions on Information Systems*, 2020, 38(2): 1-28.

[10] Sarwar B, Karypis G, Konstan J, et al. Item-Based Collaborative Filtering Recommendation Algorithms[C]. *The tenth International Conference on World Wide Web - WWW'01*, 2001: 285–295.

[11] Hu Y F, Koren Y, Volinsky C. Collaborative Filtering for Implicit Feedback Datasets[C]. *2008 Eighth IEEE International Conference on Data Mining*, 2008: 263-272.

[12] He X N, Zhang H W, Kan M Y, et al. Fast Matrix Factorization for Online Recommendation with Implicit Feedback[EB/OL]. 2017: arXiv: 1708.05024[cs.IR]. https://arxiv.org/abs/1708.05024

[13] R. Socher, D. Chen, C. D. Manning, and A. Ng. Reasoning with Neural Tensor Networks for Knowledge Base Completion[C]. *NIPS*, 2013: 926–934.

[14] Sebastian Ruder. An Overview of Gradient Descent Optimization Algorithms[RB/OL]. 2016. ArXiv Preprint ArXiv:1609.04747

[15] Kingma D, Ba J. Adam: A Method for Stochastic Optimization[EB/OL]. 2014

[16] Bayer I, He X N, Kanagal B, et al. A Generic Coordinate Descent Framework for Learning from Implicit Feedback[EB/OL]. 2016

杨正一 于 2020 年在中国科学技术大学电子信息工程专业获得学士学位。现在中国科学技术大学信息与通信工程专业攻读硕士学位。研究领域为数据挖掘、推荐系统, 图学习等。Email: yangzhy@mail.ustc.edu.cn

吴宾 于 2020 年在郑州大学软件工程专业获得博士学位。现任郑州大学信息工程学院助理教授。研究领域为机器学习、推荐系统、社会网络分析等。Email: wubin7019088@gmail.com

王翔 于 2019 年在新加坡国立大学计算机学院获得博士学位。现任新加坡国立大学 NExT++中心博士后研究员。研究领域为推荐系统、图学习与推理、深度学习技术与可解释性人工智能等。Email: xiangwang@u.nus.edu

何向南 于2016年在新加坡国立大学计算机学院获得博士学位。现任中国科学技术大学教授、博导。研究领域为信息检索与推荐、机器学习、因果推理等。Email: hexn@ustc.edu.cn