Session-based Recommendations with Recurrent Neural Networks

Accepted by ICLR 2016，文章链接：https://arxiv.org/abs/1511.06939

访问博客版

文章摘要

本文将 RNN 应用在推荐系统任务中，解决实际应用的传统推荐算法过度依赖用户做出的最后一个行为（点击、打分）数据，而在之前与之相关的行为数据则明显利用不足，经典的矩阵分解算法就捉襟见肘了。目前，通常采用折中的方法，即基于最近邻，推荐最相似的物品。本文认为，对于用户所有行为（称为一个 session ）进行建模，可以提高推荐的准确率。根据实际情况，对传统 RNN 进行改进，使之适合任务，从实验中验证了任务导向的模型设计的作用。

简介

实际场景中，session-based 推荐是机器学习和推荐系统这两个任务中相对难处理的问题。Session-based 强调，我们对于用户做出购买某个商品的决策（如从点击商品 A 页面上的链接，跳转至其他商品的页面，最终停留在商品 B 的页面，从而决定购买 B 的行为）是一个过程（ session ），而不是传统推荐系统认为的，仅仅是该用户与他购买的商品的交互数据。Session-based Recommendation 还会对用户的点击数据进行建模，即便是没有发生购买行为的商品。（注：这些数据大多隐藏在用户的浏览历史和浏览器 cookies 中，可能涉及到一些隐私问题）

现在的推荐系统和相关算法，数据来自用户的最后一次点击或是选择行为，包括但不限于诸多特征模型、最近邻方法。近些年深度学习能力之强大，可以让我们学习一些非结构化数据中的知识。RNN 的出现，让我们有能力处理序列化信息。本文认为，RNN 能提高 session-based 推荐系统的性能。根据任务导向，对 RNN 进行改造，使之能处理用户的访问行为序列，并设计了一个 **新的排序损失函数** 来训练、更新模型的参数。

这项工作可以类比 RNN 在 NLP 任务中的应用：RNN 的初始状态是用户第一个浏览的商品，随后用户的每一次浏览和点击，都会作为序列的下一个输入，并且更新 RNN 中的隐状态，我们可以根据这个隐状态对用户进行候选推荐，可以认为，推荐系统给出本次推荐的结果与用户之前的所有点击行为都相关，推荐系统推荐的商品与用户实际的点击的差异即为在当前输入下的损失。本文的另一个小贡献在于，设计的 **排序损失函数** 可以减少商品数量空间过于巨大带来的计算开销。

相关工作

Session-based 推荐

推荐系统的许多工作都只关注用户的最终的购买、点击行为，把这些数据输入模型中学习用户、商品的特征向量。那么对于一个新的用户，如果没有他的任何信息，只能推荐他点击过的商品的相似品，相似品计算是通过其他已知用户的交互行为数据间接计算的（ 如 item-based neighborhood filtering），这样的方法虽然简单，但是也行之有效，在实际应用中的效率也令人满意。

马尔科夫决策过程则有些不同，问题形式化描述为一个四元组：$\langle S, A, Rwd, tr \rangle$，其中 S 表示状态集合，A 表示状态之上的行为集合，Rwd 是一个奖励函数，tr 是状态转换函数。在推荐系统中，行为很自然被定义为推荐或不推荐。那么只需要对用户的决策过程进行抽象，很容易把 session-based 推荐转化为一个马尔科夫决策过程。但是有一个问题，那就是随着对用户的各种可能的行为预测，状态空间将变大得难以控制（大概是商品数的几何增长）。

还有一种广义的分解模型，对用户从开始浏览网页到最终做出决策的过程（即为 session ）中的每一个事件加总在一起，通过建立两个语义空间的物品表示，一是物品本身的特征表示，另一个则是物品在该 session 中的特征。对 session 中涉及物品的特征向量取平均，就得到了这个 session 的特征，以 session 为单位构建了 session 的特征，再用于后续用户推荐的任务。然而，这种方法并没有考虑到用户的这个 session 中浏览物品的顺序，就像 NLP 任务中，丢失了句子中词的顺序信息。文中没有解释后面怎么做，我的猜想是，把这个 session 特征作为用户喜欢的物品的特征，推荐和这个特征向量相似的物品。

深度学习与推荐系统

第一个将深度学习方法应用到推荐系统中来的，是使用 受限玻尔兹曼机 对用户和商品进行建模，通过协同过滤的方法进行推荐。此外，各种深度学习方法也被用来从用户、商品相关的非结构化辅助信息中抽取特征，刻画更为精确的用户、商品特征向量，结合协同过滤模型，进一步提升推荐系统的性能。

模型：RNN 与推荐系统

从 RNN 的原型到 GRU 和 LSTM 等变体，RNN 体现出了对于序列化信息较强的处理能力，并且在多种任务上成为 state-of-the-art。为了适应 session-based recommendation 这一任务，对经典的 GRU 做了一些改进：

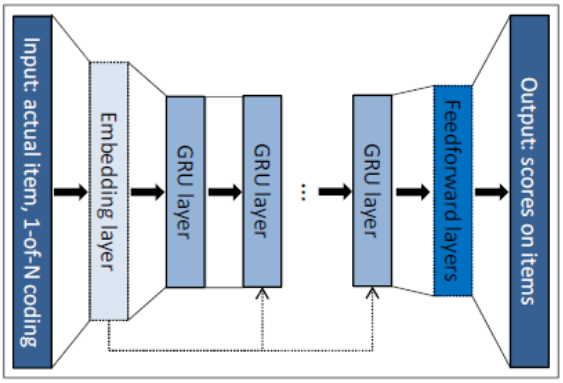

Figure 1: General architecture of the network. Processing of one event of the event stream at once.

在每一个 GRU 层，根据物品的特征向量（如 one-hot 编码）和前一个状态计算当前状态的隐向量。前一个 GRU 层的输出作为后一个 GRU 层的输入。经过多层 GRU 的作用，最后一个 GRU 得到的隐状态向量通过一个前馈层预测对商品的得分。由于 RNN 并不是直接适用推荐系统的任务，还做了如下改进。

Session 并行 Mini-Batches 训练

在处理 NLP 任务时，RNN 会有滑动窗口覆盖句子中的词，把窗口中的片段组合成一个个的 mini batch。在推荐系统中，主要有两点不同：

1. 每一个 session 的长度不固定，这一现象较句子长度的差异更明显，session 的长度最小可以是 2，最长可能有几百；
2. 我们的目的是抽象出整个 session 在时间变化的过程中的特征，那么对一个 session 的切分就会变得没有意义。

因此，采用训练过程中采用 session parallel mini-batches。首先，对所有的 session 进行排序；根据 mini-batch 的 size 取对应的 session 数，每个 session 中第一个物品即为 mini-batch 的第一批输入，其预期的输出对应第二个物品，依次类推。当其中某个 session 结束时，就选择一个可用的 session 接在后面。每个 session 都是独立的，需要注意的是，每个 session 的最后一个状态不作为输入。具体的流程见下图：

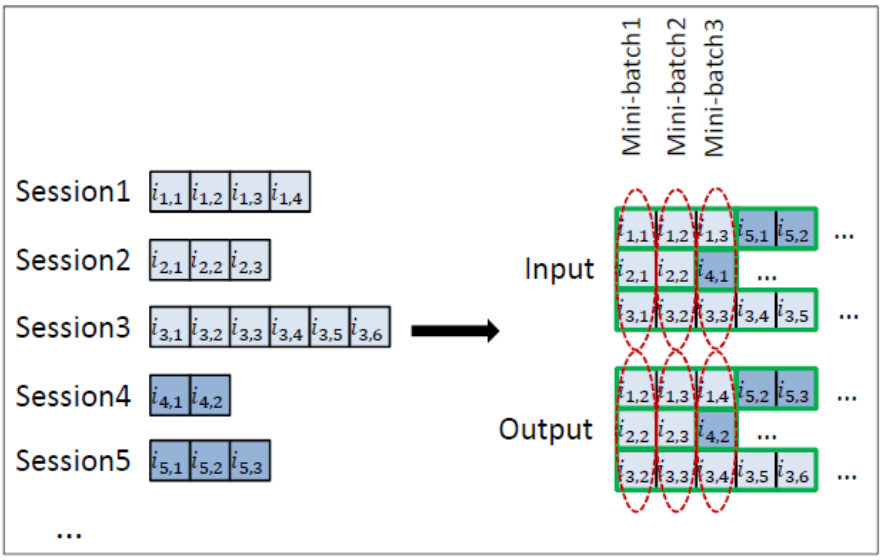

Figure 2: Session-parallel mini-batch creation

输出采样

模型的最终输出，需要对所有可能的物品计算预测得分，当物品的数量极大时，在实际应用中的计算和时间开销就很大了。一定程度上缓解这种规模问题，提出了一种采样方式，在一个较小的子集上计算物品的预测得分。具体做法是在输出层加入许多的负样本，在最小化损失的过程中，只需要确保正确的物品的排序更为靠前。

这里有一种很自然的解释，那些没有被采样到的样本，可以看做该用户也不知道这些物品的存在。然而有一个问题，实际上用户知道某个商品，出于不喜欢而选择 **没有点击** 的行为的概率也是非常小的。考虑到越是热门的物品就越可能被普通用户知悉，所以负采样与物品的流行度成正比。

为了进一步加快计算性能，文中负样本的生成是来自 mini-batch 中的其他物品，这也符合根据流行度采样的准则，毕竟在 mini-batch 中出现的，至少还是有一定热度的。

排序损失

得到对物品的预测打分，就进入推荐的环节。推荐系统的实践证明，抽象为一个排序问题要比分类问题（买或不买的二分类）更好。计算排序损失主要有三种：

1. Point-wise，只需要确保目标物品的得分较高，或者排序靠前
2. Pair-wise，确保（1个）正样本的得分高于（1个对应的）负样本，或者排序前于负样本
3. List-wise，正样本要比其他所有样本的得分高，即排序靠前

实验证明，pair-wise 定义的损失具有更好的性能，主要有以下两种：

1. BPR，Bayesian Personalized Ranking，定义损失 $L_s = -\frac{1}{N_s} \cdot \sum_{j=1}^{N_s} \log(\sigma(\hat{r}_{s,i} - \hat{r}_{s,j}))$

   其中 N_s 是采样的数量，i 表示预期物品，j 则是负样本。

2. TOP1，对于正样本 i 和负样本 j，正样本的相对排序（化归到 [0,1] 区间） 是 $\frac{1}{N_s} \cdot \sum_{j=1}^{N_s} \mathbb{I}(\hat{r}_{s,j} > \hat{r}_{s,i})$

   最后的损失 $L_s = \frac{1}{N_s} \cdot \sum_{j=1}^{N_s} \sigma(\hat{r}_{s,j} > \hat{r}_{s,i}) + \sigma(\hat{r}_{s,j}^2)$

实验

一共有两个数据集，第一个 RSC15 是来自 2015 年 RecSys Challenge，是用户在电商网站上的浏览和购买物品数据；第二个数据集是 VIDEO，包含了 youtube 网站的用户观看视频数据，统计信息如下：

Datasets	sessions	events	items
RSC15	7,966,257	31,637,239	37,483
VIDEO	about 3 million	about 13 million	about 330,000

实验设置上，由于 RSC15 的商品数不算很多，可以做全局的排序，VIDEO 上则不行，所以设置了负采样子集规模为 30,000。

评价方式：从这些数据中随机选择一部分作为测试集，对于这些 session 预测每一个指标为 Recall@20 和 MRR@20。

基线实验

1. POP，选择最流行、火热的物品进行推荐；
2. S-POP，选择当前 session 中最火热的物品进行推荐，这在可以 **重复浏览** 的情况下具有很好的效果；
3. Item-KNN，基于物品的最近邻，即选择最相似的物品进行推荐；
4. BPR-MF，矩阵分解，即协同过滤算法，损失使用BPR。

基线实验 结果如下：

Table 1: Recall@20 and MRR@20 using the baseline methods

Baseline	RSC15		VIDEO	
	Recall@20	MRR@20	Recall@20	MRR@20
POP	0.0050	0.0012	0.0499	0.0117
S-POP	0.2672	0.1775	0.1301	0.0863
Item-KNN	0.5065	0.2048	0.5508	0.3381
BPR-MF	0.2574	0.0618	0.0692	0.0374

可见，我们常用的矩阵分解算法（BPR-MF）在实际问题中表现较差，反而是简单的 Item-KNN 取得较好的效果。

模型参数与实验结果

涉及到一些超参： GRU 的层数，GRU unit的个数，batch-size， dropout，学习率设置等。

Table 3: Recall@20 and MRR@20 for different types of a single layer of GRU, compared to the best baseline (item-KNN). Best results per dataset are highlighted.

Loss / #Units	RSC15		VIDEO	
	Recall@20	MRR@20	Recall@20	MRR@20
TOP1 100	0.5853 (+15.55%)	0.2305 (+12.58%)	0.6141 (+11.50%)	0.3511 (+3.84%)
BPR 100	0.6069 (+19.82%)	0.2407 (+17.54%)	0.5999 (+8.92%)	0.3260 (-3.56%)
Cross-entropy 100	0.6074 (+19.91%)	0.2430 (+18.65%)	0.6372 (+15.69%)	0.3720 (+10.04%)
TOP1 1000	0.6206 (+22.53%)	**0.2693 (+31.49%)**	**0.6624 (+20.27%)**	**0.3891 (+15.08%)**
BPR 1000	**0.6322 (+24.82%)**	0.2467 (+20.47%)	0.6311 (+14.58%)	0.3136 (-7.23%)
Cross-entropy 1000	0.5777 (+14.06%)	0.2153 (+5.16%)	–	–

实验结论

1. 经典RNN 和 LSTM 的效果会差一些；
2. 损失函数如果使用 cross_entropy 或 MRR，模型很难收敛（不稳定），解释是负采样在训练过程中的影响太小了，容易对正样本过拟合；前面介绍的 BPR 和 TOP1 损失性能稳定，且结果较好；
3. 多层 GRU 会带来负面效果，而GRU 的 units 多一些，效果会更好；
4. 模型训练时间，几个小时，作者说可以接受。

总结与点评

本文将 NLP 中成功的例子 RNN 应用到推荐系统上来，并且从直觉上是符合用 RNN 处理序列化信息的机器学习经验的。并且根据推荐系统任务的特征，对经典 RNN 框架进行了适应性的修改，主要体现在 mini-batch 的并行训练、输出层负采样和损失函数的设计。

作者说，本文可以作为深度学习在推荐系统中的应用和 session-based 推荐任务的基础。我认为，本文的成功之处在于根据任务的特征，将 RNN 成功转移到这项新的任务上来，还是以一种符合我们直觉的方式，针对任务的特性进行了改进。尤其是输出层负采样，给出的解释，让人觉得这样做有很强的动机和说服力。

个人认为，可以把 RNN 应用到时序有关的推荐系统任务中来，将用户的打分按照时间排序，然后用某个用户之前的打分信息去预测对下一个物品的打分。模型的测试可以把用户最后一个打分作为测试集，与矩阵分解等方法比较，数据集有MovieLens，值得尝试。