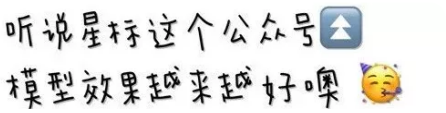
万物皆可Graph | 当推荐系统遇上图神经网络

原创 上杉翔二 NewBeeNLP 1月14日

收录于话题
#推荐搜索 7 #图网络学习 9



> NewBeeNLP原创出品
> 公众号专栏作者@上杉翔二
> 悠闲会 · 信息检索

图神经网络可以说是现在AI领域的超级宠儿。针对推荐系统的稀疏性问题，图方法还真的很适合，主要原因有下：

- 推荐系统中存在很多的图结构，如二部图，序列图，社交关系图，知识语义图等
- GNN比传统的随机游走等能有更好的表现

「PinSage」和**「EGES」**都是很好的落地实践方法，也是这篇文章的重点。不过首先来看一下对于user-item二部图的一般处理方法**「GCMC」**，通用的捕捉主要是需要处理四步，

1. 构图，包括如何采样
2. 邻居聚合，包括多少信息应该被选择
3. 信息更新，如何整合中心节点的信息
4. 最后的节点表示，然后再做下游的推荐预测任务。

> **PS！文末有我们新建立的『图神经网络』专题讨论组，名额有限，赶紧加入一起精准交流吧！**

GCMC

> - 论文：Graph Convolutional Matrix Completion
> - 地址：https://arxiv.org/pdf/1706.02263.pdf

- arxiv访问不方便的同学后台回复『0012』直接获取论文

发表在KDD2018，将user-item矩阵补全问题抽象成了二分图的连接预测问题。每种连接预测的边可以视为lable（如点击，购买，收藏等等，1-5的评分也可以），然后用二部图的方法来进行链接预测（即预测是否点击或者预测评分是多少，就变成了分类预测问题）。

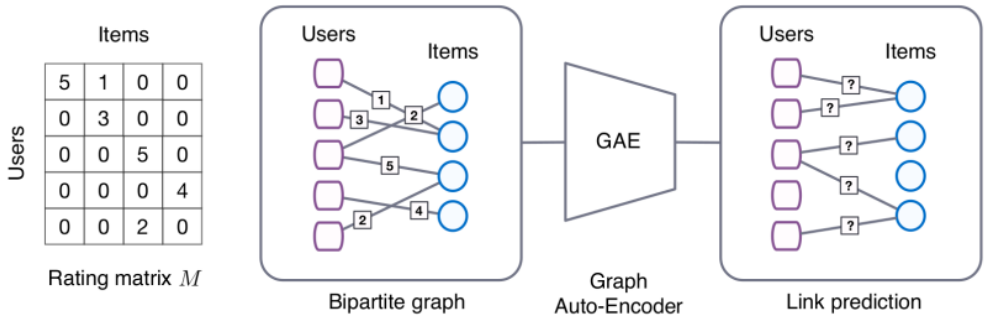


Figure 1: ***Left*****: Rating matrix M with entries that correspond to user-item interactions (ratings between 1-5) or missing observations (0).** ***Right*****: User-item interaction graph with bipartite structure. Edges correspond to interaction events, numbers on edges denote the rating a user has given to a particular item. The matrix completion task (i.e. predictions for unobserved interactions) can be cast as a link prediction problem and modeled using an end-to-end trainable graph auto-encoder.**

虽然也可以图特征提取模型和链接预测模型，不过本文具体方式上如上图，是使用图自编码器进行端到端建模。

Graph convolutional encoder

使用节点消息传递来更新模型参数，比如从item j 到 user i:

$$\mu_{j\to i,r} = \frac{1}{c_{ij}} W_r x_j^v$$

其中c是正则化，W是控制变类型的权重，x是j的特征。从user到item当然也是同样的方法，然后对每个节点都进行这样的消息传递，最后就能得到每个节点的表示（如user的表示是用一个权重W对其所有相邻节点的表示聚合得到）。

Bilinear decoder

重建链路，即把每个不同的评分等级（或者点击，购买等）看作是一类

$$P = \frac{e^{(z_i^u)^T Q_r z_j^v}}{\sum_{s=1}^{R} e^{(z_i^u)^T Q_r z_j^v}}$$

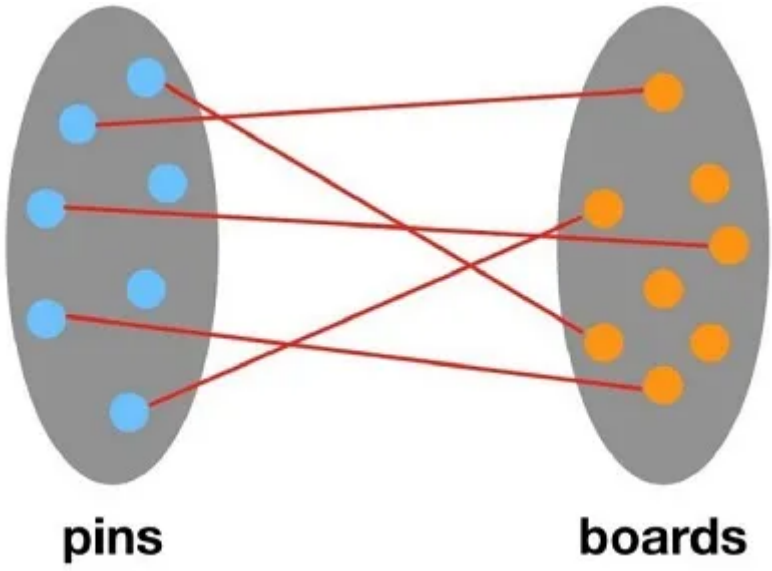
其中 z_i^u ， z_j^v 是用户，商品的表示，Q是转化矩阵，最后计算一个softma分数来预测“分类”就可以了。

GCMC使用消息传递+自编码的思想，实际上在大图应用中对每个节点做消息传递太过复杂。在实践应用中还是使用随机游走+其他，接下主要介绍PinSage，EGES。

PinSage

- 论文：Graph Convolutional Neural Networks for Web-Scale Recommender Systems
- 地址：https://arxiv.org/abs/1806.01973
- arxiv访问不方便的同学后台回复『0013』直接获取论文

PinSage，一个能够学习节点嵌入的随机游走GCN，由Pinterest公司和Stanford完成的工作，首次将图方法落地到了工业界。PinSage的理论背景是基于GraphSAGE[1]，即归纳(inductive)式的学习，直接学习聚合函数而不是固定的节点，这也是其他的图算法如GCN等等直推式(transductive)方法无法做到的，更能满足实际中的图节点是不断变化的需求（节点和关系都会不断的变化）。



目地

首先看看需求。Pinterest公司是世界上最大的图片社交分享网站，业务采用瀑布流的形式向用户展现图片（抖音也很像这种模式），并且允许用户创建和管理主题图片集合。网站上的大量图片称为pins，而用户喜欢的图片集合（类似收藏夹），即称为pins钉在画板 的pinboards上。于是pins和boards就形成了如开头图片所示的二部图形式。

挖掘这种二部图的目的在哪里？分析用户兴趣，帮助用户发现和匹配他们感兴趣的图片（商品）。虽然Pinterest的数据显然都是一堆图片，但是图片节点本身的信息是无法通过CNN-based方法来解决的。如图像识别，床栏和花园栅栏都是条状的“栏”，被分为一类的概率很大，这并不能提供很多有用的信息，但是如果看看这两个图片在Graph中的位置就会发现区别很大，因为大门和花园栅栏通常会成为邻居节点，但是床和大门却很少相邻。这也就是图的优点，可以通过邻居节点的信息，位置得到更丰富的嵌入特征。

模型

模型的输入Graph是数十亿对象的web-scale graph（30亿个节点，180亿），节点是图片（需要注意的是，节点特征包括视觉特征和文本特征）。然后可以将图分成Pin和Pinboard（实际上可以看作是pin的上下文信息），依照关系可以构建二部图。

PinSage基于GraphSAGE，GraphSAGE[2]博主以前已经整理过了，所以不做过多的展开。简单来说它的核心思想就是学习聚合节点的邻居特征生成当前节点的信息的**「聚合函数」**，有了聚合函数不管图如何变化，都可以通过当前已知各个节点的特征和邻居关系，得到节点的embedding特征。

GraphSage(Graph SAmple and aggreGatE)，很重要的两步就是Sample采样和Aggregate聚合，PinSage也是一样。

首先是Convolve部分，这部分相当于GraphSage算法的聚合阶段过程，伪代码如下：

Algorithm 1: CONVOLVE

Input : Current embedding $\mathbf{z}_u$ for node u; set of neighbor embeddings $\{\mathbf{z}_v | v \in \mathcal{N}(u)\}$, set of neighbor weights $\boldsymbol{\alpha}$; symmetric vector function $\gamma(\cdot)$

Output: New embedding $\mathbf{z}_u^{\text{NEW}}$ for node u

1 $\mathbf{n}_u \leftarrow \gamma\left(\{\text{ReLU}\left(\mathbf{Q}\mathbf{h}_v + \mathbf{q}\right) \mid v \in \mathcal{N}(u)\}, \boldsymbol{\alpha}\right)$;

2 $\mathbf{z}_u^{\text{NEW}} \leftarrow \text{ReLU}\left(\mathbf{W} \cdot \text{CONCAT}(\mathbf{z}_u, \mathbf{n}_u) + \mathbf{w}\right)$;

3 $\mathbf{z}_u^{\text{NEW}} \leftarrow \mathbf{z}_u^{\text{NEW}} / \|\mathbf{z}_u^{\text{NEW}}\|_2$

输入是当下节点u的嵌入 z_u，然后得到它的邻居节点 z_e，然后主要对应伪码的1，2，3步：

1. 聚合邻居。可以看到，所有的邻居节点特征 h_z 都经过一层dense层（由Q和q参数控制，再ReLU激活），再由聚合器或池化函数 γ （如mean等）将所有邻居节点的信息聚合得到n_u

2. 更新当前节点的特征。将原特征 z_u 和 n_u 拼接后再经过一层dense层（W，w参数控制，再ReLU激活）
3. 最后归一化。直接对上一步得到的特征归一化

所以其实Convolve和GraphSage的不同之处就在于聚合邻居特征前多做了一步dense层抽象特征。

然后是minibatch对应着采样邻居部分，伪代码如下：

Algorithm 2: MINIBATCH

Input : Set of nodes $\mathcal{M} \subset \mathcal{V}$; depth parameter K;
neighborhood function $\mathcal{N} : \mathcal{V} \rightarrow 2^{\mathcal{V}}$
Output: Embeddings $\mathbf{z}_u, \forall u \in \mathcal{M}$

/* Sampling neighborhoods of minibatch nodes. */
$\mathcal{S}^{(K)} \leftarrow \mathcal{M}$;
for $k = K, \ldots, 1$ **do**
  $\mathcal{S}^{(k-1)} \leftarrow \mathcal{S}^{(k)}$;
  for $u \in \mathcal{S}^{(k)}$ **do**
    $\mathcal{S}^{(k-1)} \leftarrow \mathcal{S}^{(k-1)} \cup \mathcal{N}(u)$;
  end
end
/* Generating embeddings */
$\mathbf{h}_u^{(0)} \leftarrow \mathbf{x}_u, \forall u \in \mathcal{S}^{(0)}$;
for $k = 1, \ldots, K$ **do**
  for $u \in \mathcal{S}^{(k)}$ **do**
    $\mathcal{H} \leftarrow \left\{\mathbf{h}_v^{(k-1)}, \forall v \in \mathcal{N}(u)\right\}$;
    $\mathbf{h}_u^{(k)} \leftarrow \text{CONVOLVE}^{(k)}\left(\mathbf{h}_u^{(k-1)}, \mathcal{H}\right)$
  end
end
for $u \in \mathcal{M}$ **do**
  $\mathbf{z}_u \leftarrow \mathbf{G}_2 \cdot \text{ReLU}\left(\mathbf{G}_1 \mathbf{h}_u^{(K)} + \mathbf{g}\right)$
end

采样方法实际上就是在某个Minibatch内，以所有节点作为起始节点，然后做一个bfs去获取一定数量的邻居节点（步长固定为K的路径），最后按照逐层方式进行多层的卷积。

- 2~7行是邻居采样阶段。不沿用GraphSage的随机采样，而是使用访问数作为重要性采样。即每次从当前节点出发随机走，虽然一开始是平均的，游走很多次之后，被走到的次数越多的节点，它相对于当前节点的重要性就越高，最终选取top-t的邻居。
- 然后后面就是Convolve操作了逐层的生成节点嵌入特征。特别注意就是要经过dense之后才更新特征（伪码15-16）。

训练损失使用的是常规负采样之后，再使用的max-margin ranking loss，即最大化正例之间的相似性，同时保证与负例之间相似性小于正例间的相似性：

$$J_G(z_q, z_i) = E_{n_k \sim P_n(q)} max\{0, z_q \cdot z_{n_k} - z_q \cdot z_i + \triangle\}$$

训练技巧

- 简单负采样会导致模型分辨的粒度过粗，没针对性，特别是数据量如此大的情况下。所以增加了“hard”负样本，即根据当前节点相关的PageRank排名，选排名在2000-5000之间的items为“hard”负样本候选集，再进行随机采样，以增加训练难度。

- Multi-GPU形式，minibatch取值为512-4096不等，大的batchsize可能会导致收敛困难。所以使用warmup：在第一个epoch中将学习率线性提升到最高，后面的epoch中再逐步指数下降。

- Minibatch里具有较多样本，一个GPU无法计算。所以将这个Minibatch的图切成很多个子图，每个子图在一个GPU上，来做GPU并行的求梯度，最后再将梯度汇集起来更新参数。

- 为了大规模计算设定的两个MapReduce任务：
  - 执行聚合所有pins的嵌入特征。即把所有的pins映射到一个低纬度空间
  - 执行采样邻居特征得到board的嵌入特征，即直接通过采样邻接节点的特征来获得。向量最终都会存在数据库中供下游任务使用，实验证明这种方法在不到24小时内可以为所有30亿样本生成Embedding。

PinSage的工程启示

- 在邻接点的获取中，采用基于Random Walk的重要性采样
- Hard负样本抽取

- 基于邻接点权重的重要性池化操作
- 利用图片，文字信息构建初始特征向量
- 阶段性存储节点最新的Embedding，避免重复计算

PinSage和node2vec，DeepWalk的区别?

- node2vec，DeepWalk是无监督训练，而PinSage是有监督训练
- node2vec，DeepWalk不能用节点特征，PinSage可以
- node2vec，DeepWalk的参数和节点呈线性关系，很难用在超大型的图上

如果只用完全hard的负采样会怎么样?

模型收敛速度减半，迭代次数加倍。所以其实PinSage使用的是一种curriculum的方式，即第一轮简单负采样，帮助模型快速收敛，然后再逐步加入hard的样本，如第n轮使给负样本集合中增加n-1个负样本。直观来讲，PinSage有12亿个样本作为训练正例，每个batch500个负例，每张图又有6个hard负例。

负采样之道

我之前也有一个疑问，就是为什么不使用**「曝光未点击」**的样本呢?

这里我也是弄混了两个概念，所以就暂时整理在这篇文章里面吧。弄混的地方就是一般推荐系统的召回和排序的采样是不一样的！对于排序的效果提升才会比较讲究“真负”样本，所以一般就拿“曝光未点击”的样本，但这里反应的往往是用户的直接反馈。但是召回不一样，召回阶段不仅仅是它的候选集大而已，与排序侧需要挖掘用户可能的喜好不同（其得到的list已经相当规整了，所以某种程度曝光未点击策略只能算是个trick而已），召回需要将可能喜欢的尽可能得与其他海量不靠谱的样本分隔开（这里的候选集会是千差万别的），所以在召回侧如果只拿用户的反馈训练，将会一叶障目导致更多不靠谱的都筛选不掉。

所以随机采样或许会是一个好选择。但是这里又有两个坑：一是尽量不要随机选择，因为推荐中充斥着二八定律和热门商品等等，很容易被绑架，所以一般最好对热门的降采样，这就和word2vec很像了。二是随机选择的能力有限，无法学到细粒度的差异。

所以Hand Negative是很重要的。一方面和用户匹配程度最高的是正样本，另一方面完全差异很大的是负样本，而hard negative需要找到难度适中的，没那么相似的样本，所以才有刚刚PinSage也使用的2000-5000的这个范围，并且先简单负采样，再逐步加hard，且按照facebook EBR提到的，easy：hard维持在100：1是比较合理的。

EGES

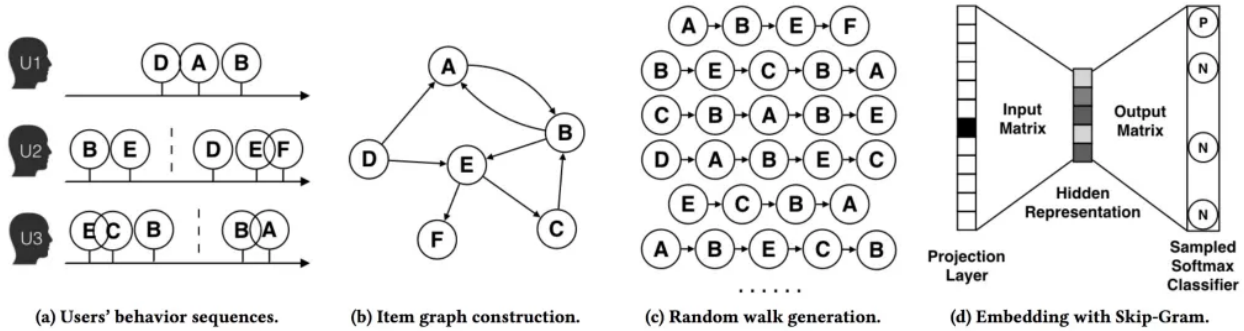


Figure 2: Overview of graph embedding in Taobao: (a) Users' behavior sequences: One session for user u1, two sessions for user u2 and u3; these sequences are used to construct the item graph; (b) The weighted directed item graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$; (c) The sequences generated by random walk in the item graph; (d) Embedding with Skip-Gram.

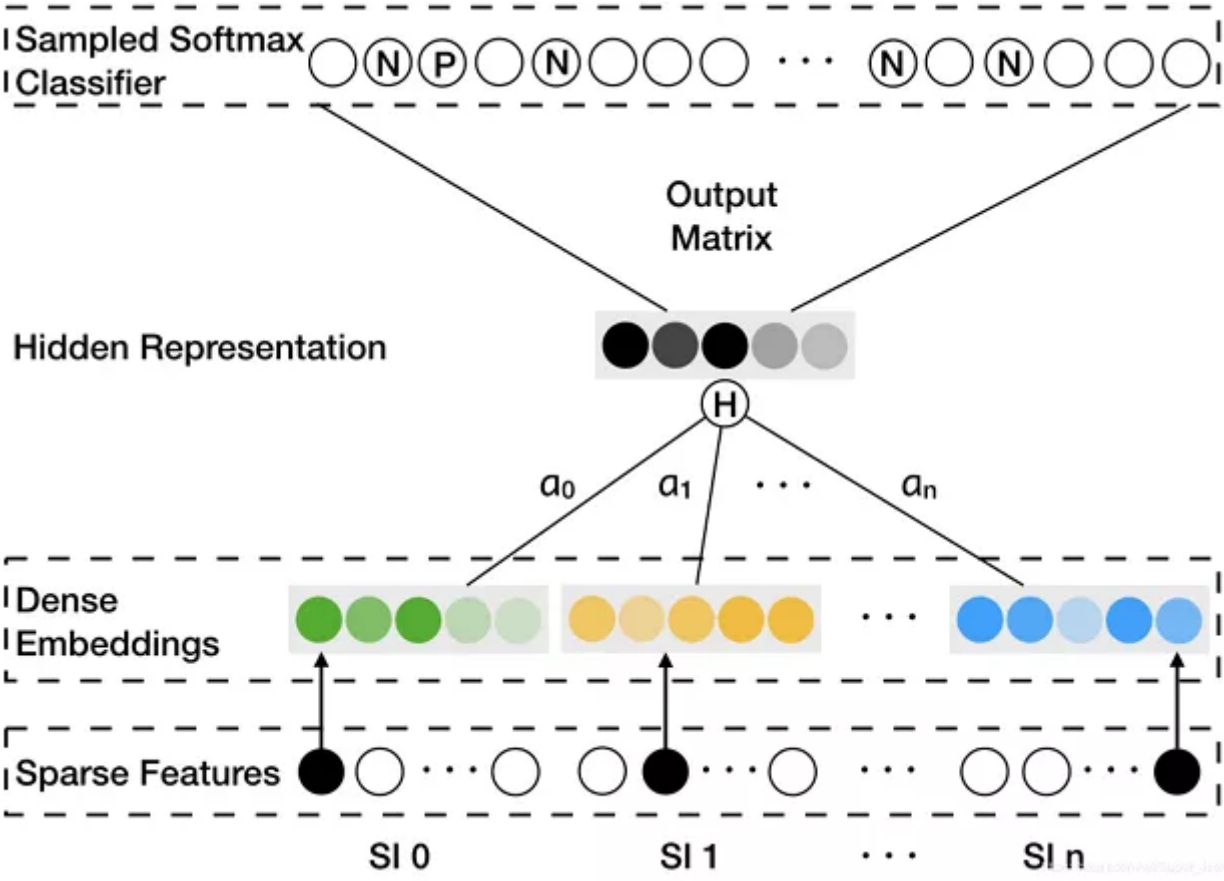
- 论文：Billion-scale Commodity Embedding for E-commerce Recommendation in Alibaba
- 地址：https://arxiv.org/abs/1803.02349
- arxiv访问不方便的同学后台回复『0014』直接获取论文

同样的阿里其实也基于GraphSAGE做过相关工作（或许GraphSAGE在工业落地上是最有优势的了），发自KDD 2018。为了解决在推荐系统中的老三样困难：可扩展性，稀疏性和冷启动。所以提出了：

- 可以根据用户的历史行为来构建商品图（如上图的a和b，根据用户的一些点击记录，依照连续点击应该存在关系，可以构造出如b一样的item graph），并学习图中所有商品的嵌入（如图c的带权随机游走采样，用deepwalk[3]思想基于skip-gram训练embedding）。这一步就是作者描述的Base Graph Embedding（BGE）。

- 为了减轻稀疏性和冷启动问题，将边信息合并到图嵌入框架中。即增加商品的属性，如品牌，价格等。然后就升级成了Graph Embedding with Side information (GES)，并且对不同的side Information加权得到Enhanced Graph Embedding with Side information (EGES)。

插一句这个历史行为构图其实就是session-based会话推荐了，这个下一篇文章会继续详细介绍。EGES模型图如下：



看图可知，就是把特征one-hot之后Embedding，再用注意力算一次权重H之后concat所以的特征，最后一层dense就得到了商品的特征。EGES只是把side Information带权加入进来，即把每个item的特征变丰富了。之后还是用deepwalk[4]思想采样变“句子”，基于skip-gram训练embedding，DeepWalk博主整理过了就不说了吧，最后也是一个负采样训练的常规操作。

SR-GNN

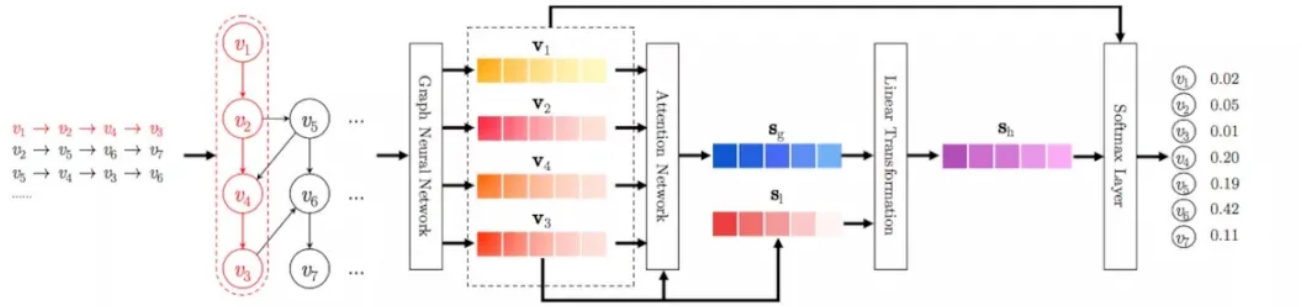


Figure 1: The workflow of the proposed SR-GNN method. We model all session sequences as session graphs. Then, each session graph is proceeded one by one and the resulting node vectors can be obtained through a gated graph neural network. After that, each session is represented as the combination of the global preference and current interests of this session using an attention net. Finally, we predict the probability of each item that will appear to be the next-click one for each session.

会话序列推荐的图应用，发自AAAI 2019，先放链接：

- 论文：Session-based Recommendation with Graph Neural Networks
- 地址：https://arxiv.org/abs/1811.00855
- arxiv访问不方便的同学后台回复『0015』直接获取论文

这篇文章和上一篇很像，只是不是用DeepWalk，而是GNN的。首先，会话推荐是指，对于一个用户的点击序列（即session），预测下一个要点击的物品。即输入所有的物品V={v1,v2,...,vm} ，在给定某个session为s=[v1,v2,...,vn]的用户点击序列，预测下一个要点击的物品vn+1。

现有基于会话的推荐，方法主要集中于循环神经网络和马尔可夫链，但是存在以下缺陷：

- 当一个session中用户的行为数量十分有限时，RNN很难产生好的用户特征
- 马尔可夫链非常依赖数据独立性的假设
- 同时，物品之间的转移模式在会话推荐中是十分重要，但RNN和马尔可夫过程只对相邻的两个物品的单向转移关系进行建模，而忽略了会话中其他的物品（即上下文）。

所以不如直接将会话序列建模为图结构数据，并使用图神经网络捕获复杂的项目物品item间转换，每一个会话利用注意力机制将整体偏好与当前偏好结合进行表示。同时这种方式也就不依赖用户的表示了，完全只基于会话内部的潜在向量获得Embedding，然后预测下一个点击。

Graph Neural Networks

对于构图的表示，可以看模型图的最左边红色部分，对于一连串的点击序列，就直接利用点击关系构图就ok。然后抽取这个会话图中的每个物品的Embedding向量，利用物品的Embedding向量再进行预测。

抽取序列中物品特征的GNN部分使用 Gated GNN，其计算公式为：

$$\mathbf{a}_{s,i}^{t}=\mathbf{A}_{s,i:}\left[\mathbf{v}_{1}^{t-1},\ldots,\mathbf{v}_{n}^{t-1}\right]^{\top}\mathbf{H}+\mathbf{b}, \tag{1}$$

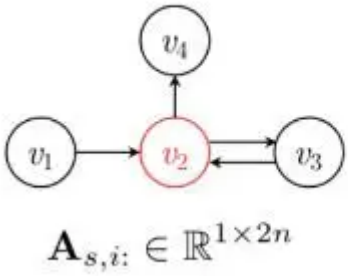
$$\mathbf{z}_{s,i}^{t}=\sigma\left(\mathbf{W}_{z}\mathbf{a}_{s,i}^{t}+\mathbf{U}_{z}\mathbf{v}_{i}^{t-1}\right), \tag{2}$$

$$\mathbf{r}_{s,i}^{t}=\sigma\left(\mathbf{W}_{r}\mathbf{a}_{s,i}^{t}+\mathbf{U}_{r}\mathbf{v}_{i}^{t-1}\right), \tag{3}$$

$$\widetilde{\mathbf{v}}_{i}^{t}=\tanh\left(\mathbf{W}_{o}\mathbf{a}_{s,i}^{t}+\mathbf{U}_{o}\left(\mathbf{r}_{s,i}^{t}\odot\mathbf{v}_{i}^{t-1}\right)\right), \tag{4}$$

$$\mathbf{v}_{i}^{t}=\left(1-\mathbf{z}_{s,i}^{t}\right)\odot\mathbf{v}_{i}^{t-1}+\mathbf{z}_{s,i}^{t}\odot\widetilde{\mathbf{v}}_{i}^{t}, \tag{5}$$

H和b是权重矩阵和偏置，v是点击物品序列。A_s 是由两个邻接矩阵，即入度和出度A(in)和A(out)矩阵拼接而成的(n，2n)的矩阵，而 $A_{s,i}$ 可以理解成矩阵的第i行(1，2n)。如下图所示：



	Outgoing edges				Incoming edges			
	1	2	3	4	1	2	3	4
1	0	1	0	0	0	0	0	0
2	0	0	1/2	1/2	1/2	0	1/2	0
3	0	1	0	0	0	1	0	0
4	0	0	0	0	0	1	0	0

$A_{s,2}$ 代表的就是红色的那一行，这样做的目的是使模型能结合出度入度以学习到更复杂的图上下文关系。至此GNN的部分实际上就结束了.....后面的四个公式就是普通的RNN了，即利用某点在图中的特征表示再RNN。

```
def get_slice(self, i):
    inputs, mask, targets = self.inputs[i], self.mask[i], self.targets[i]
    items, n_node, A, alias_inputs = [], [], [], []
    for u_input in inputs:
      n_node.append(len(np.unique(u_input)))
    max_n_node = np.max(n_node)#最大的session的item数目
    for u_input in inputs:
      node = np.unique(u_input)#unique的item
      items.append(node.tolist() + (max_n_node - len(node)) * [0])#不够的补0
      u_A = np.zeros((max_n_node, max_n_node))#user的邻接矩阵
      for i in np.arange(len(u_input) - 1):
        if u_input[i + 1] == 0: #为0说明这个session已经结束了
          break
        u = np.where(node == u_input[i])[0][0]
        v = np.where(node == u_input[i + 1])[0][0]
        u_A[u][v] = 1
      \#最终想要的邻接矩阵A是入度和出度A(in)和A(out)矩阵拼接而成的(n, 2n)的矩阵
      u_sum_in = np.sum(u_A, 0) #按0维度sum，即入度总数
      u_sum_in[np.where(u_sum_in == 0)] = 1 #防止没有某节点没有入度而除了0
      u_A_in = np.divide(u_A, u_sum_in) #平均一下
      u_sum_out = np.sum(u_A, 1) #同理按1sum，算一下出度
```

```
        u_sum_out[np.where(u_sum_out == 0)] = 1
        u_A_out = np.divide(u_A.transpose(), u_sum_out) #需要转置一下
        u_A = np.concatenate([u_A_in, u_A_out]).transpose() #最后拼接两者
        A.append(u_A)
        alias_inputs.append([np.where(node == i)[0][0] for i in u_input])
    return alias_inputs, A, items, mask, targets
```

拿到A之后再做GNN，按上面图中的公式就可以了。

```
def GNNCell(self, A, hidden):
    \#因为A是入度和出度两个矩阵拼接得到的，所以要分0:A.shape[1]和A.shape[1]: 2 * A.s
    input_in = torch.matmul(A[:, :, :A.shape[1]], self.linear_edge_in(hidder
    input_out = torch.matmul(A[:, :, A.shape[1]: 2 * A.shape[1]], self.linea
    inputs = torch.cat([input_in, input_out], 2)#然后再拼接
    gi = F.linear(inputs, self.w_ih, self.b_ih)#输入门
    gh = F.linear(hidden, self.w_hh, self.b_hh)#记忆门
    i_r, i_i, i_n = gi.chunk(3, 2)#沿2维度分3块，因为线性变换这三门是一起做的
    h_r, h_i, h_n = gh.chunk(3, 2)
    \#三个gate
    resetgate = torch.sigmoid(i_r + h_r)
    inputgate = torch.sigmoid(i_i + h_i)
    newgate = torch.tanh(i_n + resetgate * h_n)
    hy = newgate + inputgate * (hidden - newgate)
    return hy
```

Attention策略

得到item的特征向量后（模型图中的彩色条），应用一个注意力机制。有一点不同的是作者认为在当前的会话序列中，最后一个物品是非常重要的，所以单独将它作为 s_1，然后计算其他的物品与最后一个物品之间的相关性，再加权就得到了 s_g 以考虑到全局信息：

$$\alpha_i = q^T \sigma(W_1 v_n + W_2 v_i + c)$$

$$s_g = \sum_{i=1}^{n} \alpha_i v_i$$

接着将得到的 s_1 和 s_g 连接，输入到一个线性层

$$s_h = W_3[s_1; s_g]$$

最后使用 s_h 和每个物品的embedding进行内积计算，再softmax得到最终每个物品的点击率，最后交叉熵得到损失函数：

$$L(y') = -\sum_{i=1}^{m}(y_i log p(y_i') + (1-y_i')log(1-p(x)))$$

```
def compute_scores(self, hidden, mask):
    \#计算全局和局部，ht是最后一个item，作者认为它代表短期十分重要
    ht = hidden[torch.arange(mask.shape[0]).long(), torch.sum(mask, 1) - 1]
    q1 = self.linear_one(ht).view(ht.shape[0], 1, ht.shape[1])  # batch_size
    q2 = self.linear_two(hidden)  # batch_size x seq_length x latent_size
    alpha = self.linear_three(torch.sigmoid(q1 + q2)) #计算全局注意力权重
    \# 不算被mask的部分的sum来代表全局的特征a
    a = torch.sum(alpha * hidden * mask.view(mask.shape[0], -1, 1).float(),
    if not self.nonhybrid: #globe+local
      a = self.linear_transform(torch.cat([a, ht], 1))#将局部和全局s1和sg拼接
    b = self.embedding.weight[1:]  # n_nodes x latent_size
    scores = torch.matmul(a, b.transpose(1, 0)) #再做一次特征变换
    return scores
```

完整的代码中文逐行笔记：https://github.com/nakaizura/Source-Code-Notebook/tree/master/SR-GNN

总结

当然只要有关系的地方就能构图，就能抽图特征，特别是最近Graph这么火....万物皆可Graph。比如有人的地方就能有社交关系等等，深入挖掘属性等等，最近也有很多论文是针对属性中的异构信息[5]进行挖掘，下一篇博文再整理。

一起交流

想和你一起学习进步！我们新建立了『**图神经网络**』专题讨论组，欢迎感兴趣的同学加入一起交流。为防止小广告造成信息骚扰，麻烦添加我的微信，手动邀请你