GRU4rec

SESSION-BASED RECOMMENDATIONS WITH RECURRENT NEURAL NETWORKS，通过循环神经网络来进行会话推荐。

先来理解一下Session-Based Recommendation的定义。它的中文翻译是基于会话的推荐，我们可以理解为从进入一个app直到退出这一过程中，根据你的行为变化所发生的推荐；也可以理解为根据你**较短时间内**的行为序列发生的推荐，这时session不一定是从进入app到离开，比如airbnb的论文中，只要前后两次的点击不超过30min，都算做同一个session。

1、模型介绍

1.1 背景介绍

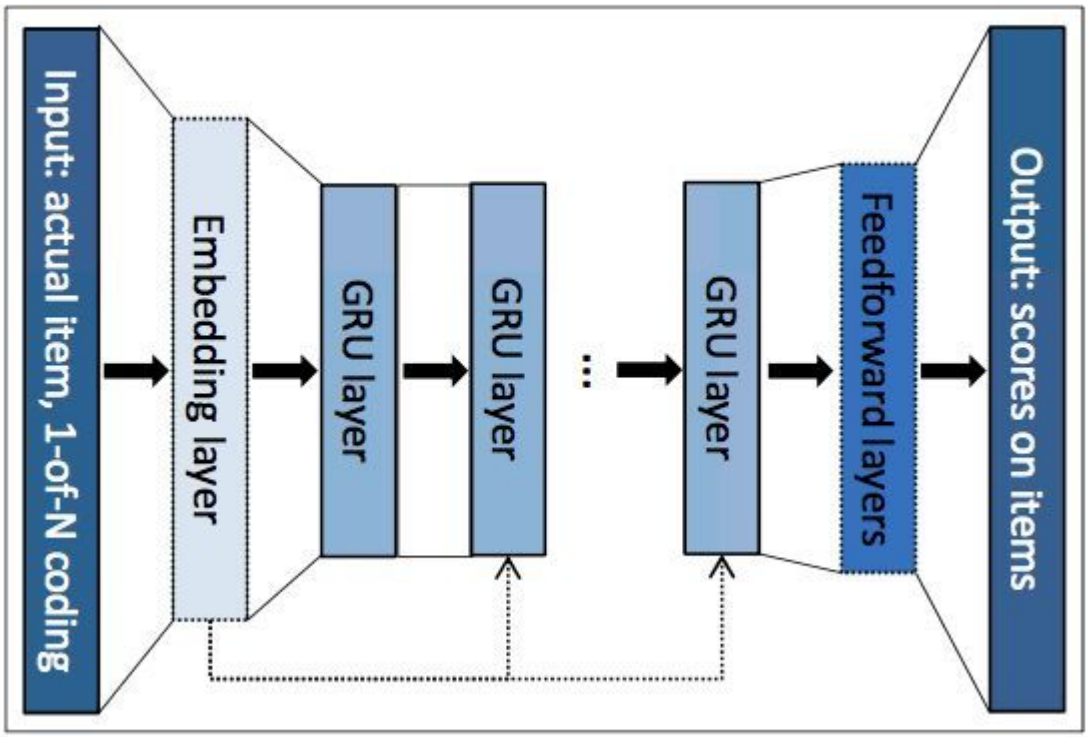
在本文出现之前（2016年），基于会话的推荐方法，主要有**基于物品的协同过滤**和**基于马尔可夫决策过程**的方法。

基于物品的协同过滤，需要维护一张物品的相似度矩阵，当用户在一个session中点击了某一个物品时，基于相似度矩阵得到相似的物品推荐给用户。这种方法简单有效，并被广泛应用，但是这种方法只把用户上一次的点击考虑进去，而没有把前面多次的点击都考虑进去（论文里这么说，不过我认为可以按比例混合多次点击的推荐结果吧）。

基于马尔可夫决策过程的推荐方法，也就是强化学习方法，其主要学习的是状态转移概率，即点击了物品A之后，下一次点击的物品是B的概率，并基于这个状态转移概率进行推荐。这样的缺陷主要是随着物品的增加，建模所有的可能的点击序列是十分困难的（可能论文年代比较久远，现在的话我们应该可以使用DQN等方法了）。

1.2 基于RNN的会话推荐

回到正题，文中提出使用基于RNN的方法来进行基于会话的推荐，其结构图如下：



模型的结构很简单，对于一个Session中的点击序列x=[x1,x2,x3...xr-1,xr]，依次将x1、x2,...,xr-1输入到模型中，预测下一个被点击的是哪一个Item。

首先，序列中的每一个物品xt被转换为one-hot，随后转换成其对应的embedding，经过N层GRU单元后，经过一个全联接层得到下一次每个物品被点击的概率。

物品数量如果过多的话，模型输出的维度过多，计算量会十分庞大，因此在实践中一般采取**负采样**的方法(sampled softmax)。论文采用了取巧的方法来减少采样需要的计算量，即选取了同一个batch中其他 sequence 下一个点击的 item作为负样本，用这些正负样本来训练整个神经网络。

采用minibatch负采样的好处是什么呢？其中有一点是，用户未点击某个商品其实并不能够说明他不喜欢这个商品，而可能只是他没见过罢了。所以，对于popular的商品，他很有可能见过，但是并不喜欢，所以更应该作为负样本。mini-batch sampling，则是一种基于popularity的采样方法。

1.4 Rank Loss

这里，论文提出了两种pair-wise的损失函数，分别为BPR和TOP1。

BPR BPR损失，对比了正样本和每个负样本的点击概率值，其计算公式如下：

$$-\frac{1}{N_S}\cdot\sum_{j=1}^{N_S}\log\left(\sigma\left(\hat{r}_{s,i}-\hat{r}_{s,j}\right)\right),$$

其中，i代表的是正样本，j代表的是负样本，若正样本的点击概率大于负样本的点击概率，这样损失会比较小，若正样本的点击概率小于负样本，损失会比较大。

TOP1 第二个损失函数感觉和第一个损失函数差不多，只不过对负样本的点击概率增加了正则项(使负样本的得分尽可能接近于0)，同时sigmoid之后没有再取log：

$$L_s = \frac{1}{N_S} \cdot \sum_{j=1}^{N_S} \sigma\left(\hat{r}_{s,j} - \hat{r}_{s,i}\right) + \sigma\left(\hat{r}_{s,j}^2\right)$$

1.6 关于损失函数的讨论

为什么使用pair-wise的损失函数，要比point-wise的损失函数更好呢？这主要还是看场景吧。比如在电商领域、外卖点餐的时候，我们可能很多东西都喜欢，但是只会挑选一个最喜欢的物品进行点击或者购买。这种情况下并不是一个非黑即白的classification问题，只是说相对于某个物品，我们更喜欢另一个物品，这时候更多的是体现用户对于不同物品的一个相对**偏好关系**，此时使用pair-wise的损失函数效果可能会好一点。

在广告领域，一般情况下用户只会展示一个广告，用户点击了就是点击了，没点击就是没点击，我们可以把它当作非黑即白的classification问题，使用point-wise的损失函数就可以了。

不过还是要提一点，相对于使用point-wise的损失函数，使用pair-wise的损失函数，我们需要采集更多的数据，如果在数据量不是十分充足的情况下，point-wise的损失函数也许是更合适的选择。