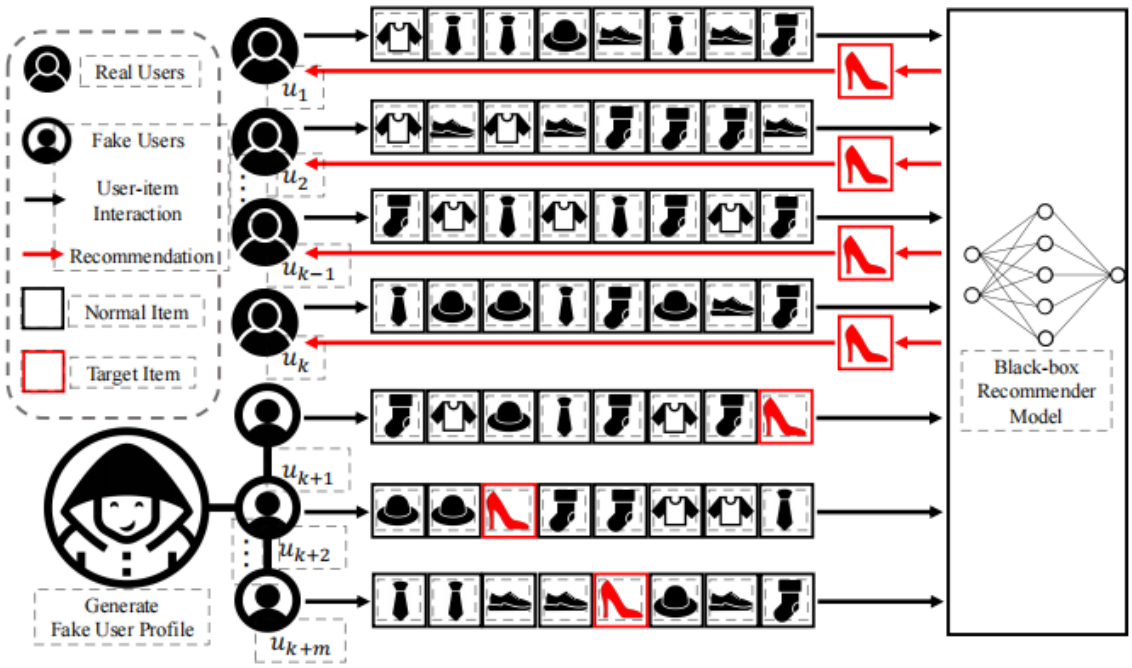
一、Poisoning Self-supervised Learning Based Sequential Recommendations

1.研究动机

在计算机视觉领域，SSL 已被证明容易受到中毒攻击，而基于 SSL 的推荐系统中的中毒攻击尚未被研究。

2.方法

推荐系统的中毒攻击可分为两种： 非针对性攻击和针对性攻击。非针对性攻击目标是降低系统性能，而本文则是后者:通过有毒数据，使得特定item更易被推荐/不推荐。



本文方法部分可概括如下：通过gan生成假的样本序列，并通过对比学习进行自监督预训练。而推荐系统的自监督预训练与CV略有不同，CV中通过对图片裁剪、旋转、颜色变换、模糊等方式形成样本对，在推荐系统中则通过mask部分item形成正负样本，同一序列不同方式的mask后组成正样本，反之则为负样本对。在本文的Poisoning Attack中将gan生成的假序列与正序列当成相似对（正样本），优化loss。

这里疑惑的点在于，gan的生成无法保证生成的序列一定包含目标item，而part c中对此做出了方法阐述:通过生成序列与目标序列计算loss并梯度更新（例如，生成的[1,2,3,4,5],给的标签是[5,5,5,5,5],这样在优化时便更易生成包含5的序列），使得生成的fake序列包含更多的目标item。这里也有个很明显的问题，这个任务太过简单容易过拟合，因此作者在给定的标签中加入了概率阈值p（这里猜测应该是借鉴了dropout的处理方式）

本文在下游任务中，验证自己Poisoning Attack预训练的参数有效性，混入1%数据进行下游任务fintune的效果验证，与原始数据相比预测的概率提升了一个数量级。

Dataset	Attack	*Sample 99*						*Full*					
		HR@1	HR@5	NDCG@5	HR@10	NDCG@10	MRR	HR@5	NDCG@5	HR@10	NDCG@10	HR@20	NDCG@20
S^3-Rec													
Beauty	Pure	0.0001	0.0138	0.0059	0.0575	0.0198	0.0348	0	0	0	0	0	0
	Random	0.0004	0.0175	0.0080	0.0553	0.0200	0.0334	0	0	0	0	0	0
	Popular	0.0033	0.0918	0.0449	0.1566	0.0658	0.0636	0	0	0	0	0	0
	RecommPoison	0.0003	0.0209	0.0092	0.0935	0.0322	0.0380	0	0	0	0	0	0
	Our Attack	**0.2559**	**0.7542**	**0.5132**	**0.9142**	**0.5658**	**0.4615**	**0.0343**	**0.0223**	**0.0558**	**0.0293**	**0.0877**	**0.0372**
Sports	Pure	0.0005	0.0144	0.0067	0.0533	0.0190	0.0328	0	0	0	0	0	0
	Random	0.0042	0.0402	0.0214	0.0940	0.0385	0.0459	0	0	0	0	0	0
	Popular	0.0013	0.0216	0.0108	0.0621	0.0236	0.0346	0	0	0	0	0	0
	RecommPoison	0.0010	0.0140	0.0070	0.0367	0.0142	0.0288	0	0	0	0	0	0
	Our Attack	**0.3326**	**0.7642**	**0.5566**	**0.9320**	**0.6113**	**0.5159**	**0.0779**	**0.0539**	**0.1125**	**0.0651**	**0.1573**	**0.0764**
Toys	Pure	0.0020	0.0263	0.0131	0.0702	0.0270	0.0372	0	0	0	0	0	0
	Random	0.0037	0.0287	0.0159	0.0654	0.0275	0.0373	0	0	0	0	0.0002	0.0001
	Popular	0.0044	0.0297	0.0170	0.0537	0.0246	0.0352	0	0	0.0001	0	0.0001	0
	RecommPoison	0.0020	0.0188	0.0102	0.0403	0.0170	0.0307	0	0	0	0	0.0001	0
	Our Attack	**0.1791**	**0.6263**	**0.4042**	**0.8668**	**0.4829**	**0.3733**	**0.0244**	**0.0150**	**0.0436**	**0.0211**	**0.0804**	**0.0303**
Yelp	Pure	0.0017	0.0286	0.0141	0.1030	0.0376	0.0413	0	0	0	0	0	0
	Random	0.0253	0.0702	0.0466	0.1667	0.0773	0.0804	0.0220	0.0172	0.0258	0.0184	0.0295	0.0193
	Popular	0.1538	0.3888	0.2752	0.5184	0.3169	0.2769	0.0126	0.0071	0.0275	0.0118	0.0581	0.0195
	RecommPoison												
	Our Attack	**0.2073**	**0.4949**	**0.3531**	**0.7407**	**0.4321**	**0.3558**	**0.0445**	**0.0310**	**0.0758**	**0.0410**	**0.1450**	**0.0583**

3.贡献点

将中毒攻击引入推荐学习ssl，并充分验证可行性和有效性。

二、Towards Multi-Interest Pre-training with Sparse Capsule Network

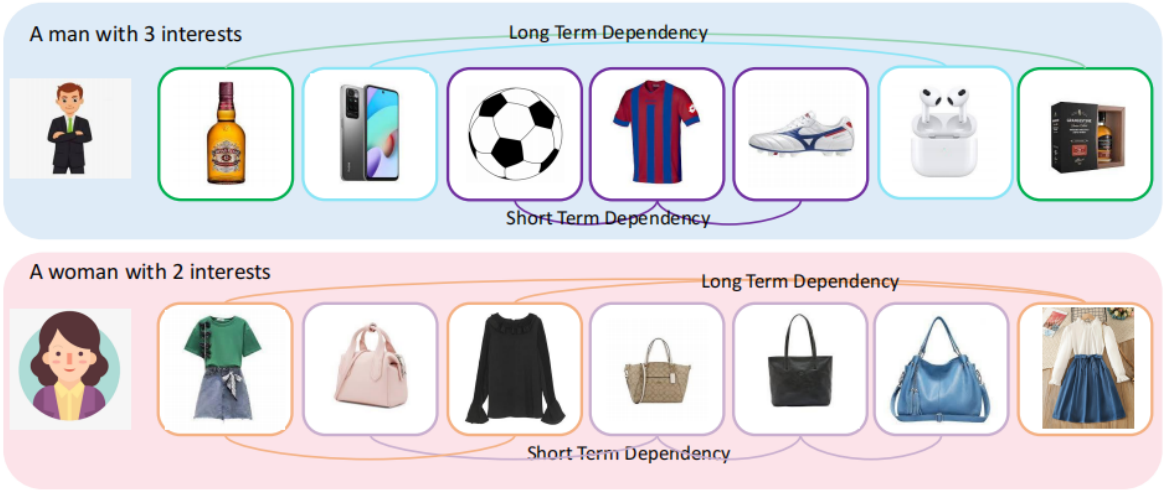
1.研究动机

用户兴趣具有动态、上下文相关等特性，且在不同域中表现复杂，常见的预训练将每个用户表示为一个固定的向量，无法捕捉用户的不同兴趣，本文研究目标是设计面向多兴趣预训练的胶囊网络。

文中对研究的多兴趣预训练框架提出了以下目标：

①对通用空间中对多样化、上下文相关和时间性的用户兴趣进行建模。文中表述现有的预训练模型通常采用类似于 GPT 的自回归架构，这可能会阻碍上下文信息建模以实现普遍的用户理解。这里我的理解是，GPT的预训练是单向推理生成，无法双向建模，例如将上文item与下文item建立双向的全局上下文关系。

②设计适用于多兴趣学习下游任务的预训练pretext task。



2.方法

提出的多兴趣预训练框架预训练阶段包含两部分： 文本感知项目嵌入（理解为序列文本的特征建模）以及稀疏兴趣胶囊网络

文本感知项目嵌入包括三部分：

①文本编码器，该部分加载BERT的预训练参数，并且该部分参数冻结，仅用作特征提取。

②MOE模块，用于将不同域的语义知识进行统一，这里原理不太熟悉，有待学习。通过代码可以看到，实质上就是将输入进行全连接,然后cat之后加噪声取均值（感觉类似于Transformer的多头注意力机制中多头multi head的处理，通过全连接，不同的head关注不同的方面）。

```
def forward(self, x):
    gates = self.noisy_top_k_gating(x, self.training)  # (B, n_E)
    expert_outputs = [self.experts[i](x).unsqueeze(-2) for i in range(self.n_exps)]  # [(B, 1, D)]
    expert_outputs = torch.cat(expert_outputs, dim=-2)
    multiple_outputs = gates.unsqueeze(-1) * expert_outputs
    return multiple_outputs.sum(dim=-2)
```

③Transformer编码器模块。对语义知识进一步编码，这里是主要的参数学习部分。

稀疏兴趣胶囊网络主要流程如下：生成基向量矩阵以构建通用兴趣空间，对于给定用户的每个项目h,计算其与每个基向量相似度，激活兴趣胶囊，确定用户兴趣数量并初始化动态路由耦合系数。后续通过全连接层获取潜在兴趣特征，经动态路由计算胶囊输出，作为用户兴趣表示。

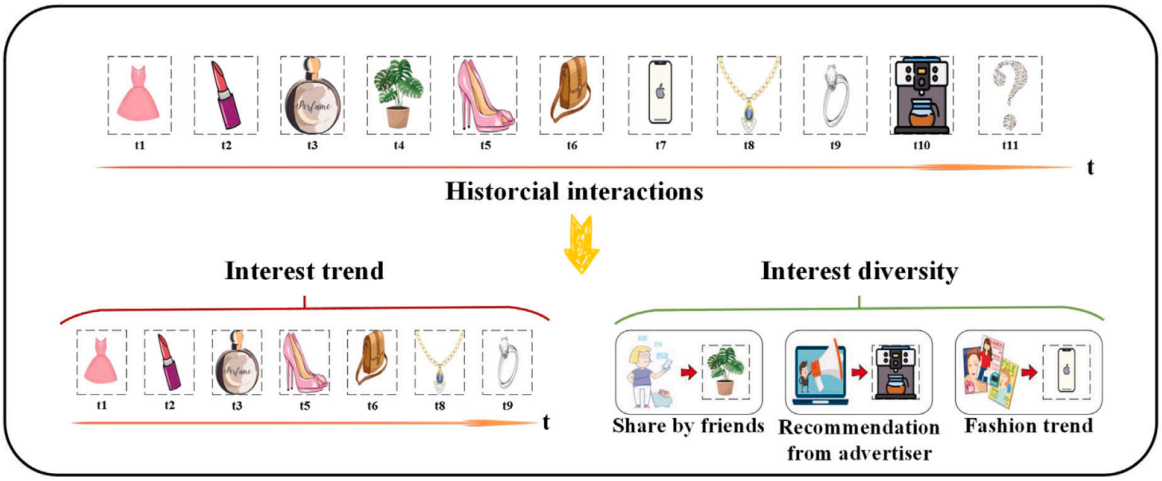
这一块实现逻辑相较其他部分比较复杂，不是特别理解，后续应单独阅读胶囊网络相关论文，补全知识。

模型的预训练遵循常见的对比学习范式，微调时冻结编码器参数，只训练解码器（本人自监督实验中也是fintune时冻结编码器，只训练解码器）。

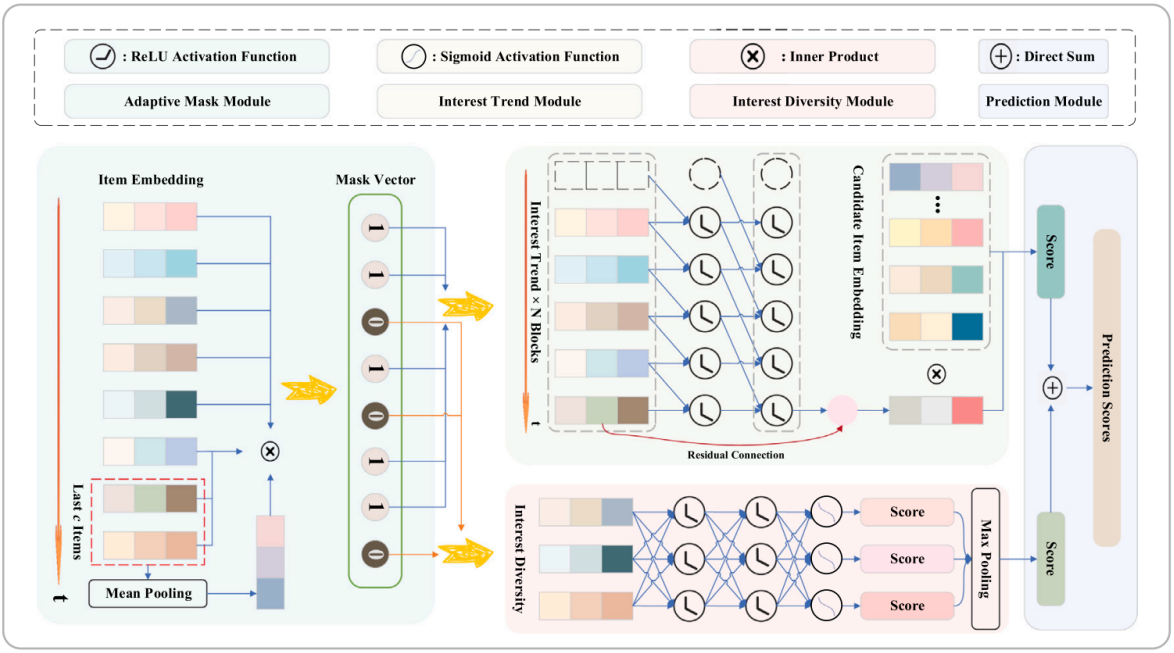
三、Disentangle interest trend and diversity for sequential recommendation

1.研究动机

作者总结现有方法存在问题：一是未区分兴趣趋势和兴趣多样性，同时建模会导致性能下降；二是在兴趣预测中消除item多样性会造成信息损失，因为这些项可能代表用户潜在兴趣。因此，本文研究目标为将兴趣趋势与兴趣多样性解耦，以提升推荐效果。作者指出**兴趣趋势与兴趣多样性的目标在某种程度上相互冲突**，这里本人较为疑惑：这里的结论似乎文中没有实验做支撑，不知道是不是已有研究验证了这一结论，有待进一步学习研究。



2.方法



因此本文的做法总结如下：将用户的历史item从兴趣趋势和兴趣多样性两个方面进行建模，各自得出预测score并进行加权从而得出最终结果。

本文较为核心的点在于如何将序列拆分为 兴趣趋势序列 与 兴趣多样性序列。这里可以看到，作者取最近的c项作为最近的趋势倾向，然后计算每一项与最近c项的余弦相似度（也就是a在b上的投影，值越接近1越相似，类似于transformer的q×k），相似度与设定的阈值比较，大于阈值该处mask就是1，以此来获取兴趣趋势序列。而兴趣多样性序列此处没说怎么分类，可能剩下的都作为了兴趣多样性序列。

$$\mathbf{p} = \frac{1}{c} \sum_{j=l-c+1}^{l} \mathbf{W}_s \mathbf{x}_{i_j}$$

$$r_j = \text{cosine}(\mathbf{x}_j, \mathbf{p}) = \frac{\mathbf{x}_{i_j}^T \cdot \mathbf{p}}{\left\| \mathbf{x}_{i_j} \right\| \|\mathbf{p}\|}$$

$$m_j = \begin{cases} 1 & , if \ r_j \geq \theta_m \\ 0 & , \text{otherwise} \end{cases}$$

然后对于兴趣趋势序列用TCN编码，兴趣多样性序列用简单的MLP对item编码并获取最大值（使用常规的sigmoid函数），对两个模块的输出score使用简单的线性插值函数加权输出结果。

Experimental results (%) on the four public datasets. We highlight the best results and the second-best results in boldface and underlined respectively.

Datasets	Metric	GRU4Rec	Caser	SASRec	BERT4Rec	ComiRec	TiMiRec	DSAN	FMLP	Teddy
Beauty	HR@5	1.0112	1.6188	3.2688	2.1326	2.0495	1.9044	2.0288	3.7192	**5.5778***
	HR@10	1.9370	2.8166	6.2648	3.7160	4.4545	3.3434	2.9965	6.6843	**7.4713***
	HR@20	3.8531	4.4048	8.9791	5.7922	7.6968	5.1674	4.3621	9.0901	**9.7754***
	NDCG@5	0.6084	0.9758	2.3989	1.3207	1.0503	1.2438	1.2914	1.8406	**4.0994***
	NDCG@10	0.9029	1.3602	3.2305	1.8291	1.8306	1.7044	1.6013	2.7991	**4.7100***
	NDCG@20	1.3804	1.7595	3.6536	2.3541	2.6451	2.1627	1.7483	4.3670	**5.2884***
Toys	HR@5	1.1009	0.9622	4.5333	1.9260	2.3026	1.1631	2.3853	3.5917	**5.9575***
	HR@10	1.8553	1.8317	6.5496	2.9312	4.2901	1.8169	0.9546	6.4665	**7.5024***
	HR@20	3.1827	2.9500	9.2263	4.5889	6.9357	2.7156	3.2311	**9.9358**	9.5637
	NDCG@5	0.6983	0.5707	3.0105	1.1630	1.1571	0.7051	1.1208	1.7948	**4.4524***
	NDCG@10	0.9396	0.8510	3.7533	1.4870	1.7953	0.9123	1.1321	2.7229	**4.9529***
	NDCG@20	1.2724	1.1293	4.3323	1.9038	2.4631	1.1374	1.5341	3.5935	**5.4700***
Sports	HR@5	0.7030	0.9724	1.7166	1.0943	1.0366	0.8294	0.7762	1.7700	**2.8218***
	HR@10	1.3725	1.7346	3.2472	1.9643	2.4815	1.5502	1.2387	3.1752	**3.9955***
	HR@20	2.6636	2.8860	5.3177	3.3960	4.5992	2.6782	1.9309	4.9165	**5.5420***
	NDCG@5	0.4086	0.5676	0.8948	0.6806	0.5085	0.4991	0.8434	0.9052	**2.0355***
	NDCG@10	0.6230	0.8112	1.3832	0.9595	0.9742	0.7308	1.2659	1.3572	**2.4136***
	NDCG@20	0.9475	1.1000	1.9035	1.3181	1.5058	1.0140	0.7714	1.7952	**2.8014***
Steam	HR@5	3.1210	3.4505	4.4205	4.9281	3.3003	5.4290	5.5943	4.9883	**5.7950***
	HR@10	5.6733	6.5863	7.7550	8.1075	6.6944	8.8356	9.1324	8.7471	**9.6035***
	HR@20	9.9166	11.3021	12.7055	12.8597	11.7563	13.8937	14.3108	14.3264	**15.0762***
	NDCG@5	1.8673	1.8551	2.7109	3.1168	1.6713	3.4937	3.5719	3.0543	**3.6897***
	NDCG@10	2.6859	2.8603	3.7808	4.1369	2.7608	4.5865	4.7072	4.2587	**4.9129***
	NDCG@20	3.7507	4.0441	5.0248	5.3303	4.0327	5.8567	6.0085	5.6611	**6.2884***

从结果来看，效果还是很明显的。特别是在排序指标NDCG上取得较大的提升，也证明了兴趣趋势模块的有效性。

四、Spectral and Geometric Spaces Representation Regularization for Multi-Modal Sequential Recommendation

1.研究动机

多模态信息在对序列推荐任务有较大帮助，但多模态推荐存在计算成本高和表示退化问题。本文拟通过轻量化 Transformer 模块和可扩展注意力机制，融合多模态信息。本文也是在Transformer的计算上进行研究，优化其在序列推荐任务上的性能。

$$\text{Attention}(Q,K,V)=\text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

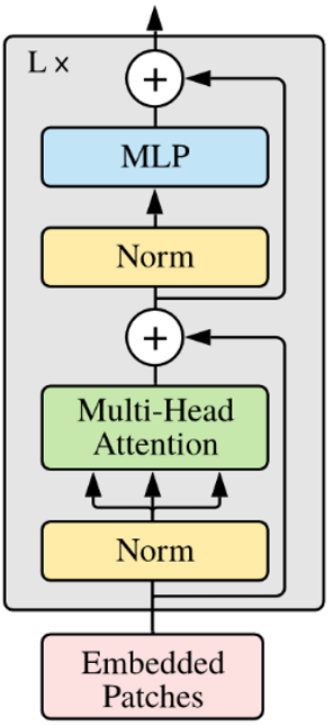
$$\text{MultiHead}(Q,K,V)=\text{Concat}(\text{head}_1,...,\text{head}_\text{h})W^O$$

$$\text{where head}_\text{i}=\text{Attention}(QW_i^Q,KW_i^K,VW_i^V)$$

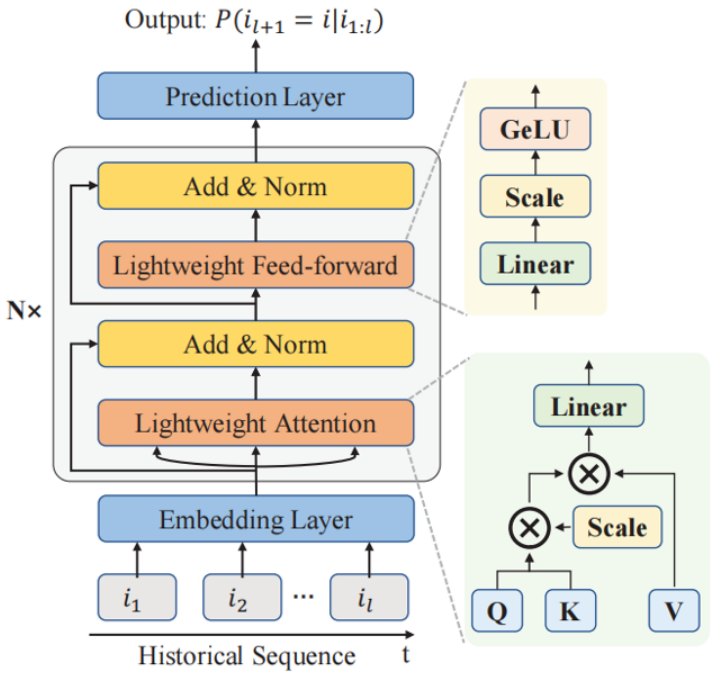
2.方法

下图把原始的Transformer与本文的做了对比，可见本文在qkv的处理上进行了改进（Transformer的主要可学习参数与运算都发生在 输入全连接成qkv以及qkv的矩阵相乘上）。

Tranformer block



this paper

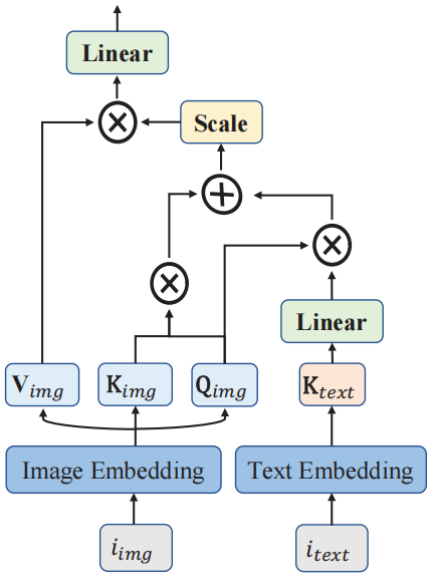


按照文中表述，用两个可学习的重缩放向量代替了自注意力组件中 query、key、value 的线性投影操作以及前馈网络 （FFN） 层中的第二次线性变换。这里没有怎么阅读paper,更加感兴趣于代码的实际运算处理上，本文核心的改动逻辑如下：

```
corr = torch.matmul(q, k.transpose(-2, -1)) / math.sqrt(q.size(-1))
corr = corr * self.scale_w.unsqueeze(-1).unsqueeze(0)
```

这里说实话并不是很理解，corr * self.scale_w.unsqueeze(-1).unsqueeze(0) 仅仅对QK所得矩阵用一个可学习向量进行逐个元素的点乘，理论上不会影响参数的计算量，为什么最终会大幅节省gpu内存占用率？此处有待结合实验深入学习

本文Transformer另一个魔改的点 融合多模态的交叉注意力机制，印象中在其他论文（特别是多模态相关的）中有过相似的处理，此处不做赘述，本质上都是多模信息全连接 所得qkv之间的矩阵运算



五、DiffuRec: A Diffusion Model for Sequential Recommendation

1.研究动机

将扩散模型应用于序列推荐

文章从四个部分阐述了，为什么研究把扩散模型应用于序列推荐?

①一个item可能具有多个类别，单个向量不好做编码。

②用户的兴趣多样性不适合静态项目表示

③用户当前的兴趣是不确定的，更适合用将兴趣建模为分布（扩散模型的加噪去噪训练目标就是学习input的数据分布）

④涉及目标item的交互需要优化（这里不理解,有待学习）

2.方法

这里的扩散模型主体流程与传统的扩散模型类似，正向逐步加噪并采样，反向过程拟合噪声并最小化损失。

CV领域的扩散模型多基于Unet架构，而本文使用的是Transformer架构做主干网络。之前认为扩散模型正向加噪与反向去噪的过程需要Unet类型的对称结构，读了这篇论文之后查了下现在的扩散模型很多都是基于Transformer的，之前对扩散模型理解有误。

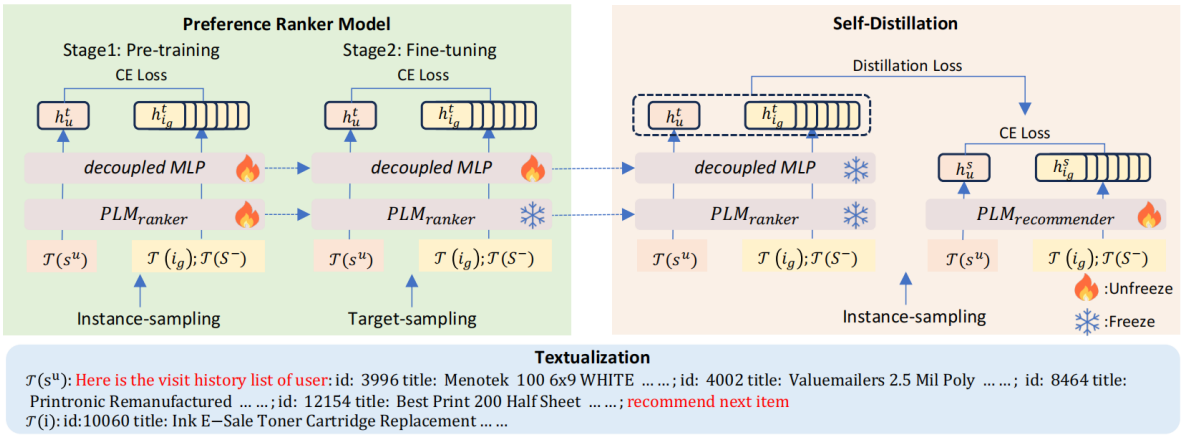
对于损失计算，文中提出扩散模型的标准目标函数不适合用于推荐任务，item的embedding不是连续分布的，因此在扩散阶段利用交叉熵损失进行模型优化（通过顺序推荐的内积计算两个向量之间的相关性）

六、TEXT CAN BE FAIR: Mitigating Popularity Bias with PLMs by Learning Relative Preference

1.研究动机

推荐模型常依赖用户行为训练，而用户行为稀疏且二元，导致模型易受流行度偏差影响，倾向推荐热门物品，使长尾物品曝光不足，形成恶性循环。现有的利用物品文本语义虽对长尾item有帮助，但仍受流行度偏差影响(作者在主流方法上进行长尾问题的实验，证明当前主流方法均表现不佳)

2.方法



用预训练语言模型构建 PLMRec 作为偏好排序器和推荐模型的骨干网络

在偏好排序器模型中分为两个阶段训练：第一阶段用实例采样预训练，捕捉数据集中的常见的排序模式；第二阶段用目标采样微调，冻结语言模型参数仅更新 MLP，获取偏好排序器模型参数。可以理解为一阶段训练编码器能力，二阶段训练解码器能力。在下面的蒸馏中，编码器、解码器参数直接使用，进行知识迁移。

在自蒸馏阶段，将排序器模型视为 RLHF 中的奖励模型，为推荐模型训练提供额外监督信号，避免长尾物品训练不足。作者提出了三种蒸馏策略：成对蒸馏（学习随机负样本与真实物品间的相对偏好，计算成对蒸馏损失）、硬蒸馏（直接匹配排序器模型中排名最高的物品进行监督，计算硬蒸馏损失）、软蒸馏（推荐模型匹配真实物品和负样本的相对偏好排名，通过最小化两个概率分布的 KL 散度计算软蒸馏损失）。实验证明成对蒸馏和软蒸馏在去除流行度偏差影响表现较好