初探图文检索任务报告

日期：2023 年 5 月 7 日

摘　要

本次任务基于 VSE++ 模型与 flickr30K 数据集，实现了一个简易的图文检索系统。本报告从任务整体实现方面，介绍了系统数据预处理、模型骨架、损失函数以及训练方式等，并训练了四个模型，在召回率等指标上依次比较分析。最后就本次的任务做了总结反思。

关键词： 图文检索，双塔模型

1　任务简述

本次任务为对图文检索领域的初步探索，要求使用 flickr30K 数据集实现一个图文检索任务的训练和评估。实现内容可以分为：图文检索数据集的 DataLoader 实现、VSE(Dual Encoder) 方法的实现以及训练和测试脚本三方面任务。

2　任务实现

2.1　完成总览

本次任务，我在论文 VSE++[2] 的代码基础上改进实现。如图 1 所示使用 6 个脚本，其中 train.py 作为训练主脚本，param.py 为多个脚本提供运行时输入的参数。vocab.py 生成词汇表，data.py 读取数据包包装为 batch，为 train.py 提供数据。随后 train.py 将数据传递给 model.py 中训练模型，并使用 evaluation.py 得到对模型的评价指标，在 train.py 中作为日志输出。

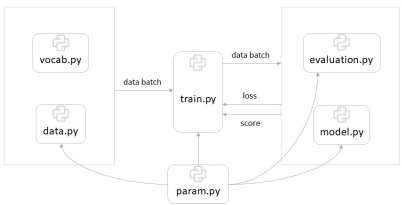

图 1: 代码实现结构

2.2　DataLoader 实现

2.2.1　数据集分析

Flickr30K 数据集是一个图像描述生成工作中常用的数据集，其包含 31000 张来自 Flickr 的图片，每张图片有 5 条由人类注释员提供的参考句子。该数据集被分割为训练集 29K，验证集 1K，测试集 1K。

下载之后，数据包含一个 dataset.json 文件和 images 文件夹，前者包含对每张图片的文件名、句子描述以及 id 等信息，后者为所有图片文件的文件夹。dataset.json 中数据形式如图 2 所示，该数据包含两个 keys：images 和 dataset，其中我们需要的数据存储在 images 中。images 中每个图片对应数据有 5 条句子以及图片的 id、图片名称、完整句子以及该项数据的划分类型。dataset 对应的 value 为该数据集的名称 flickr30K。

```
dataset.json中的数据形式:
    keys: ['images' , 'datset']

    images: {
        'sentids': [5615, 5616, 5617, 5618, 5619],
        'imgid': 1123,
        'sentences': [
            {'tokens': ['three', 'teenagers', 'are', 'carrying', 'w
            'raw': 'Three teenagers are carrying wood down a street
            {'tokens': ['a', 'man', 'in', 'a', 'blue', 'and', 'whit
            'raw': 'A man in a blue and white t-shirt trips while c
            {'tokens': ['men', 'are', 'struggling', 'to', 'carry',
            'raw': 'Men are struggling to carry wood down the stree
            {'tokens': ['these', 'three', 'boys', 'are', 'trying',
            'raw': 'These three boys are trying to carry some wood.
            {'tokens': ['three', 'boys', 'carry', 'wooden', 'racks'
            'raw': 'Three boys carry wooden racks down a sidewalk.'
        ],
        'split': 'train',
        'filename': '1352231156.jpg'
    }

    dataset: 'flickr30K'
```

图 2: dataset.json 数据形式

2.2.2 图片的预处理

该部分，总共包括 Resize、Normalize 和 Random Augmentation 三种处理步骤。**Resize** 操作用于调整图像大小，便于将图像数据组合成批次进行训练的同时，还可以加快计算速度并减少内存占用。**Normalize** 操作用于对图像数据进行归一化，从而有助于模型的训练。**Random Augmentation** 操作则是一种通过对图像进行一系列随机变换来增加训练数据集大小的技术。这种技术可以通过减少模型对某些属性的依赖来提高模型的泛化能力。该部分的实现均使用 torchvision.transforms 模块实现。对于训练集图片依次做 Resize、Random Augmentation 和 Normalize 操作，验证集和测试集则仅做 Resize 操作。对于训练集图片的 Random Augmentation，除了论文 [2] 代码中使用的 RandomResizedCrop(指定尺寸随机裁剪) 和 RandomHorizontalFlip(随机水平翻转) 我还尝试了直接使用在 ImageNet 数据集上学到的 AutoAugment 策略——AutoAugment()[1]。

2.2.3 文本的预处理

该部分首先需要从文本数据中抽取一个词汇表。虽然 dataset.json 数据中提供了对于一个句子的 tokens 分词，但是数据类型整体为 string 类型，不便于直接使用。因此如论文 [2] 代码中一致使用 nltk 库进行分词。后续对分词后的单词进行统计，小于一定数量的单词不添加如词汇表，最后再将标识符'<unk>'，'<start>'，'<end>'，'<pad>(id 对应为 0)' 添加进词汇表完成构建。在准备 Dataset 中，对于每一个句子描述，我再次使用 nltk 分词后再句子前后加上'<start>' 和'<end>' 标识符，并通过构建好的词汇表将所有 token 转换位词汇表对应 id。

2.2.4 get item 方法的实现

该部分主要在 FlickrDataset 类的 __getitem__() 方法中实现，根据索引返回对应图文数据对。此处需要先在 FlickrDataset 类的 __init__() 中便利构造图片和文本的 id 对——ids，其中每一张图片可以和其对应的 5 条描述语句形成 5 个 id 对。后续在 __getitem__() 方法中可以使用传入的索引直接从 ids 中取出图片 id 和句子描述 id。后续在 dataset.json 中即可使用此两 id 取出描述文本 caption 和图片的文件名。

2.2.5 collate_fn 方法的实现

该部分获取一个 batch_size 的 __getitem__() 方法返回的文本和图片数据对，并包装为 batch。其中需要注意的即为上一次任务中学长提到的，对数据先按照文本长度逆序排序，后根据一个 batch 中最大长的句子填充 <pad>(0)。

2.3 VSE 实现

2.3.1 图片编码器

该部分使用预训练模型 VGG19[3]，替换模型 classifier 层的最后一个线性层为 (4096 , embed_size) 将图片信息映射到能与文本信息对齐的向量空间。在训练中设置 VGG19[3] 除 classifier 的前 15 层不更新参数[2]。

2.3.2 文本编码器

该部分我同论文 [2] 代码一致，使用 GRU 网络，分别测试了一层 GRU 网络与两层 GRU 并添加 linear 层映射两组实验，在验证集上的结果显示仅使用一层 GRU 效果更优秀。在该部分的实现中，比较重要的是如上次学长提到的，使用 pad_padded_sequence 和 pad_packed_sequence 包装文本信息，并使用 torch.gather 取出句子最后一个单词对应的隐变量，避免对句子过多的 padding 冲淡句子含义。

2.3.3 Hinge Loss 的计算

在一个 batch 中，通过余弦相似度计算得到 (batch_size , batch_size) 的相似度矩阵；。由 batch 的构造方式得知，对角线上的图片和文本一定匹配。由此可以如图 3 所示计算出其余各列或者行与正样本相似度的差值。如图中所示，纵向为不同图片，横向为不同文本描述。在以图搜文时，如图中左下角所示，将正样本向 Caption 的方向拓展，并与相似度矩阵做差；以文搜图时，如图右下角所示，将正样

[1] 默认使用 ImageNet 数据集上学到的增强方式

[2] 设置更新 VGG19 网络所有参数时，需要耗费显存约 20G，较一般的 7G 左右增加了 13G，且在一张 3090 显卡上同样 batch_size 数训练时长是原本的 5 倍

本向 Image 的方向拓展，同样与相似度做差。随后再结合 Hinge Loss 公式：

$$l_{hinge}(T_i, I_i) = max(0, m + S(T_i, I_{i'}) - S(T_i, i_i))+ \\ max(0, m + S(T_{i'}, I_i) - S(T_i, i_i)) \quad (1)$$

计算得出 Hinge Loss。需要注意的是，如论文提到，为了提升计算效率，本处使用在每个 batch 中寻找到 hardest negatives。另一个需要注意的是：由于整体 flickr30k 数据中同一图片可以对应 5 个文本描述，可能在一个 batch 中出现同一张图片的多个文本描述，如此在 batch 中找到的 hardest negatives 很可能为原本匹配的图文对，可能会导致一定的学习退步。但是，对于本次使用的 flickr30k 数据集，抽取到的 batch 中有多个正样本的概率为 $\frac{C_5^2+C_5^3+C_5^4+C_5^5}{C_{1455000}^{batch\_size}}$，基本趋近于 0，可以忽略不记。

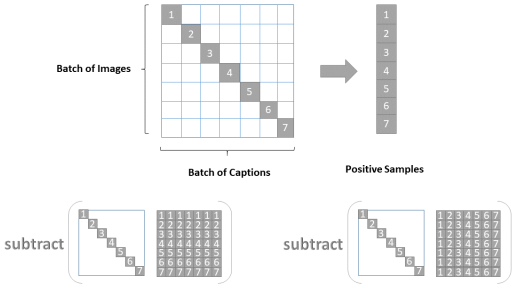

图 3: 图文匹配相似度之差

2.4 训练和测试脚本

2.4.1 recall@k 的计算

此处计算使用的数据为编码后的所有图片张量和文本张量，图片张量中连续 5 个为重复图片，且计算测试集、验证集指标时在图像增强部分使用固定的裁剪，因此图片信息张量相同。在下面两种检索的计算中，都避免重复，采用跳跃式抽取图片张量。(后续阐述中用 N 来表示不重复图片数量)

对于以图搜文：此处需要以图片张量为查询向量，每次循环抽取 5 个连续图片张量中第一个，随后再与所有 5N 个文本描述做相似度计算，得到 (1,5N) 的相似度矩阵，并得到降序排序的索引。后续在该图片对应的 5 个文本描述中用相似度最大的作为检索到的文本描述，并将其相似度排名作为检索指标 (0 到 5N-1)。

对于以文搜图：此处需要以文本张量为查询向量，每次循环抽取对应同一张图片的连续 5 个文本张量，随后与所有图片做相似度计算，得到 (5,N) 的相似度矩阵，后续进一步循环 5 次为该相似度矩阵中每一个句子对应的图片相似度降序排列，并记录匹配的图片在检索中的排名，将其作为检索指标 (0 到 5N-1)。

得到图文检索的 2 个排名指标列表后，即可统计 rank 第一为匹配查询概率、rank 前五为匹配查询概率以及 rank 前十为匹配查询概率的一共 6 个评价指标。该指标尊照前人研究指标乘以 100，将其范围固定在 (0 到 100 中)。

2.4.2 训练过程记录

学习论文 [2] 代码写法，使用 logging 作为终端输出查看当前训练效果，tensorboard_logger 生成日志文件，使用网页方式查看训练整体效果趋势。如图 4 所示，上部分为使用 logging 在终端输出当前训练 epoch、Loss 等信息，下部分为使用命令 “tensorboard –logdir=[日志文件地址] –port=[端口号]” 打开的 tensorboard 检测中 Loss 的变化曲线。

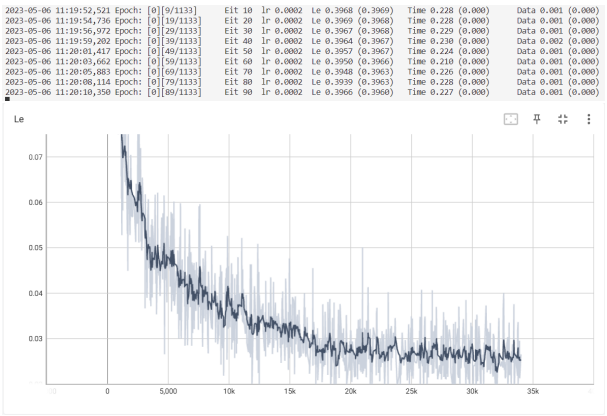

图 4: 使用日志记录训练效果

2.5 训练细节

本次训练较多参数参照论文 [2] 代码。设置 margin 为 0.2，训练 30 个 epoch，且每 15 个 epoch 学习率降低 10 倍，单词的编码维度为 300，最后对齐的文本和图片 embedding 层维度为 1024，最大下降梯度的 2，图像裁剪尺寸为 224，GRU 层数在一般情况下为 1，初始学习率为 0.0002，每计算 10 个 batch 在终端打印一次日志，每 500 个 batch 进行一次验证，图片编码方面使用预训练的 VGG19 网络，且不更新 VGG19 网络参数，模型通过在验证集合上的损失值保存最优模型。

2.6 训练结果

本次任务整体进行了四次实验，如表 1 所示的 1-VSE++、2-VSE++、3-VSE++ 与 4-VSE++。1-VSE++ 所有参数基本与论文 [2] 代码一致，文本编码部分使用一层 GRU，使用 hard_negative 计算损失，但后续所有代码均在计算 Hinge Loss 处使用 mean 而不是 sum，对于图像增强部分使用随机裁剪 (RandomResizedCrop) 和随机水平翻转 (RandomHorizontalFlip)；2-VSE++ 在前者的基础上改进，文本编码部分使用一层的双向 GRU，同样取句子最后一个隐变量后额外添加一层线性层映射到图文对齐空间；3-VSE++ 在 1-VSE++ 的图像增强部分修改，使用自动增强 (AutoAugment) 后通过随机裁剪 (RandomResizedCrop) 统一图片大小；4-VSE++ 在 1-VSE++ 的基础上使用两层 GRU 网络作为文本编码器。

表 1: 在 flickr30k 上的实验结果

Model	**Caption Retrieval**				
	R@1	**R@5**	**R@10**	**Med r**	**Mean r**
VSE++	38.6	64.6	74.6	2.0	
1-VSE++	23.5	52.2	64.4	28.6	5.0
2-VSE++	24.0	51.0	65.7	29.0	5.0
3-VSE++	22.4	50.0	62.1	33.0	5.0
4-VSE++	22.9	52.4	65.2	31.7	5.0
	Image Retrieval				
	R@1	**R@5**	**R@10**	**Med r**	**Mean r**
VSE++	26.8	54.9	66.8	4.0	
1-VSE++	20.1	45.9	58.3	35.6	7.0
2-VSE++	19.7	45.0	57.6	36.4	7.0
3-VSE++	17.8	42.9	56.4	37.6	8.0
3-VSE++	17.9	46.5	59.7	32.7	6.0

从实验结果看出与论文 [2] 中使用 MS-COCO 和 Flickr30K 数据训练的 VSE++ 相比，本次任务训练出的 3 个模型在性能方面有较大差距，尤其是从 Med r 可以看出，VSE++ 模型可以达到一半的检索都在前 2 或者前 4。而对于本次任务训练出的三个模型，2-VSE++ 相对于 1-VSE++ 在文本检索上有一定提升，这可能与增强了文本编码器的信息提取能力有关，但同时也因为参数的增加让图文对齐的难度更大[3]，因此在使用图像在大量文本中匹配时导致准确率降低。对于 3-VSE++，导致以图搜文和以文搜图准确率都降低。从原理上来说，使用图像的自动增强会给图像编码模型带来一定的泛化性提升 [1]，但或许因为实验中固定了 VGG19 网络因此并未起到期望的效果。对于 4-VSE++，虽然在两个 R@1 上评分较低，但是对于 R@5、R@10 以及 Med r、Mean r 指标得分较高，对模型的整体检索有一定提升，但这种方式与 2-VSE++ 十分相似，都增加了参数量提升文本部分的信息提取能力，但显然在与图片对齐方面效果较优。

[3] 此处的前提是图像编码部分参数锁定

3 总结与心得

先前的学习中，我对图文多模态领域有一定接触，但更多是理论上论文阅读，少有代码实现。在实现本次任务中，我一是学习到了在计算损失函数中，损失函数中最好返回一个 batch 的数据损失的平均值。这点先前单纯是看着别人如此实现，并未细想，在这次任务中我一开始与论文 [2] 代码一致使用一个 batch 的损失 sum 作为训练损失返回，但是会经常在一个 epoch 的末尾出现 Loss 的骤降，这是因为最后一个 batch 使用剩余所有数据，不一定达到 batch_size，因此计算 sum 会相对前后骤降。另一个点是第一次确实体会到 python2 代码在 python3 环境上运行的问题，部分写法不适配，并且可能出现运行结果不同的问题，同时也为了避免出现依赖冲突，所以最好对于一个项目创建一个虚拟环境。最后一个点就是从 VSE++ 的代码中学习到了很多优秀的写法，如 tensorboard 日志记录、保存 checkpoint 时一并保存当前训练 epoch、命令参数之类信息等等。

参考文献

[1] Ekin D. Cubuk et al. *AutoAugment: Learning Augmentation Policies from Data*. 2019. arXiv: 1805.09501 [cs.CV].

[2] Fartash Faghri et al. "VSE++: Improving Visual-Semantic Embeddings with Hard Negatives". In: (2018). URL: https://github.com/fartashf/vsepp.

[3] Karen Simonyan and Andrew Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. arXiv: 1409.1556 [cs.CV].