再寻图文检索任务报告

日期：2023 年 5 月 28 日

摘　要

本次任务我重构了之前的代码，并做了一些改进。在报告中首先展示了整体的代码实现框架，并在后续根据改进的内容设置了多组实验对比，包括在模型上、损失函数上以及词向量等的改进分析，最后对本次任务做了相关总结。

1　任务简述

本次任务在之前图文检索的基础上继续探究，需要完善 VSE 模型框架，并保证框架正确性、效率，并尝试从不同角度优化检索的性能。

2　任务实现

本次任务对整体代码进行了重构，尝试了加入图像端的模型微调、双向 GRU、平均池化抽取 RNN 隐变量、使用 InfoNCE 损失、使用自注意机制实现文本 Encoder、使用预训练词向量、使用不同种类模型实现 VSE 以及选择在训练中途使用 max_violation。整体代码结构如图 1 所示。

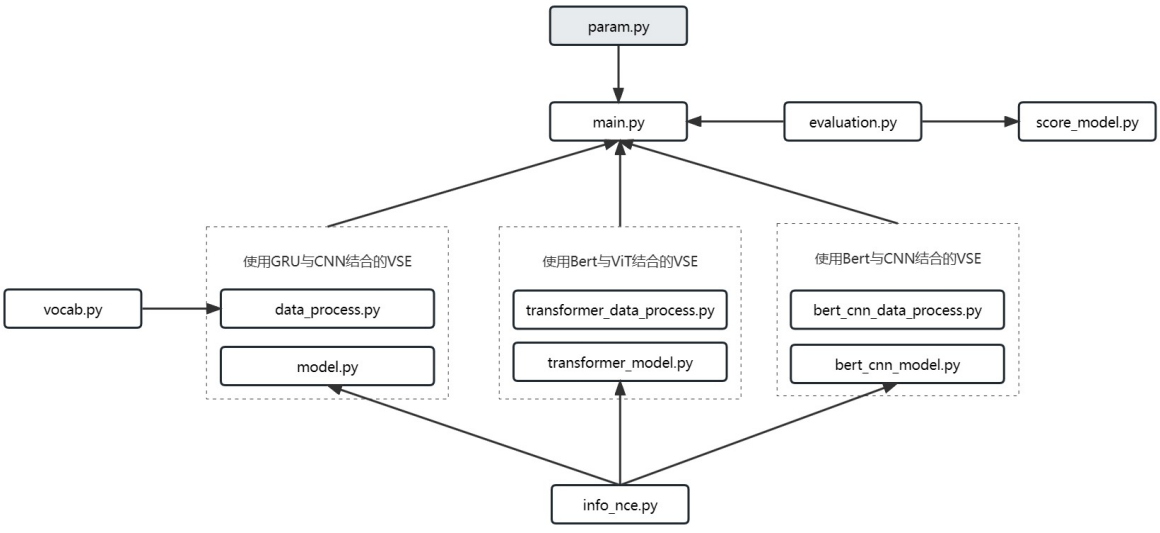

图 1: 代码实现结构。整体上 main.py 文件为主训练文件，使用模型，测评脚本 (evaluation.py) 以及参数脚本 (param.py) 完成模型训练、日志记录以及模型保存。模型构建包含模型搭建和数据预处理，但是由于 BERT、ViT 这类预训练模型所需的数据预处理较为特殊，因此将不同模型组合分成了 3 份代码，其中只有使用 GRU 的 VSE 模型需要使用 vocab.py 抽取的词汇表。evaluation.py 文件中包含测评指标的计算函数，分别用于 main.py 函数训练中评测与 score_model.py 中使用测试集评估训练完毕的模型效果。

3 改进介绍

3.1 平均池化抽取句子信息

一般使用 RNN 网络抽取句子特征都是抽取非 padding 的最后一个 token 的隐变量。但是考虑到 RNN 网络一个通病是传播越久越容易忘记之前的信息，在此做出改进，使用对所有非 padding 隐变量取 mean pool 作为抽取到的句子信息。除此之外我还设计了对于使用 attention 机制的对比实验。使用测试集评估训练的模型后，得到如表 1 所示的结果。

表 1: 平均池化方法在准确率提升方面的实验

Model	Caption Retrieval					Image Retrieval				
	R@1	R@5	R@10	Med r	Mean r	R@1	R@5	R@10	Med r	Mean r
VGG19	23.8	53.6	67.8	5.0	32.0	20.1	48.0	61.4	6.0	32.0
$VGG19_{rnnMeanPool}$	28.4	57.0	66.8	4.0	32.0	21.0	49.6	63.3	6.0	32.0
ResNet152-attention	36.9	63.5	74.8	3.0	18.0	27.4	56.5	68.1	4.0	26.0
ResNet152-$att_{rnnMeanPool}$	38.6	64.8	74.8	3.0	22.0	27.5	57.0	69.0	4.0	25.0

实验中 $VGG19$ 与 $VGG19_{rnnMeanPool}$ 模型使用 VGG19 与 2 层的 GRU 模型，不使用 max_violation，其他训练参数相同，前者抽取最后一个非 padding 的隐变量，后者使用 rnn_mean_pool 抽取所有非 padding 隐变量。$ResNet152-attention$ 和 $ResNet152-attention_{rnnMeanPool}$ 模型使用 ResNet152 作为图片端的 Encoder，使用 2 层，3 个头的多头自注意机制作为文本端 Encoder，前者抽取第一个 token 编码后的张量作为句子信息，后者使用 rnn_mean_pool 抽取所有非 padding 隐变量。对比来看，对于 rnn 网络，使用 rnn_mean_pool 可以有较大的准确率提升，但对于基于 attention 的文本 Encoder 则不会有太大的提升，这可能与注意力网络本身不存在遗忘信息的原因有关。

3.2 InfoNCE 损失

对于一个 batch_size 中的图文对样本，唯有主对角线上的图文对是正样本[1]InfoNCE 损失旨在将问题转化为分类问题，分类判断图文对是否匹配，并使用交叉熵计算损失。对于一个 batch 的数据，labels 是从经过数据处理后正样本对应的索引，inputs 为针对该图片 (文本) 在所有 batch_size 的文本 (图片) 上的相似度。实现方面使用了github上现有 InfoNCE 代码。但实际上来说，在计算交叉熵损失时，所有负样本对应的标签为 0，即对于公式：

$$CrossEntropy(inputs, labels) = -\sum_{i} Label(i) logSimScore(i)$$

$$= -(0 * logSimScore(1) + 0 * logSimScore(2) + ... + 1 * logSimScore(a) + ... = -logSimScore(a) \quad (1)$$

看起来仅使用了正样本计算 loss，但实际上因为在计算交叉熵损失时需要首先对 inputs 做 softmax 计算，在这一步的计算即使用了其余所有负样本的相似度。对于这个损失函数我并未设置专门的控制变量实验验证其效果，但是在很多实验中都默认使用该损失函数。

3.3 自注意机制实现文本 Encoder

该部分的改进为尝试使用多头自注意机制代替 rnn 网络实现文本 Encoder，从对注意力机制的印象来看，我认为其一定可以带来巨大的提升，但是与使用 rnnMeanPool 的单向 GRU 对比起来发现有略微差距，实验结果如表 2 所示。其中 $BiRGU$ 与 $BiGRU_{finetune}$ 模型使用 ResNet152 模型作为图片 Encoder，使用

[1]在此不考虑一个 batch 中出现多个相同图片的图文对的情况，因为其概率近乎为 0

表 2: 自注意机制与 GRU 的效果对比

Model	Caption Retrieval					Image Retrieval				
	R@1	R@5	R@10	Med r	Mean r	R@1	R@5	R@10	Med r	Mean r
BiGRU	41.8	70.2	80.7	2.0	13.0	32.5	63.6	74.7	3.0	17.0
Attention	39.1	66.9	79.1	2.0	16.0	31.2	60.2	71.2	3.0	23.0
BiGRU$_{finetune}$	43.5	72.4	82.1	2.0	14.0	35.3	66.6	77.3	3.0	16.0
Attention$_{finetune}$	39.1	67.6	78.1	2.0	15.0	32.6	63.7	74.9	3.0	19.0

预训练的 GoogleNews 词向量，使用 InfoNCE 损失函数，使用 rnn_mean_pool 抽取所有非 padding 隐变量，不同的是前者将 ResNet152 模型锁住，后者将 ResNet152 模型加入微调。*Attention* 与 $Attention_{finetune}$ 模型则同样使用预训练的 GoogleNews 词向量，使用 InfoNCE 损失函数，都并未使用 rnn_mean_pool 抽取所有非 padding 隐变量，同样前者不微调 ResNet152，后者微调。

从结果上来看其实使用多头注意力机制并未带来提升，甚至效果并不如使用 BiGRU 使用平均池化抽取信息。但其实本次实验设置并不完善，还有不使用平均池化的 BiGRU 以及使用平均池化的多头注意力机制两组实验未设置。

3.4 使用 GoogleNews 词向量

该处实现在使用 GRU 模型的 model.py 中，下载 GoogleNews300 词向量后为 TextEncoder 中的 embedding 层赋值，并且保存赋值后的 embedding，便于下一次创建模型时跳过对 embedding 层的赋值过程，首次加载 GoogleNews300 词向量并为 embedding 层赋值大约需要 1 分钟左右。但是对于使用 GoogleNews 词向量是否能提升模型准确性，我此处也并未设置专门的控制变量的实验，表 3 中呈现了三组具有参考价值的模型结果。其中三组实验均使用 InfoNCE 损失函数，均对于 GRU 或者 BiGRU 网络使用平均池化，其余设置仅为模型不同 (ResNet101 与 ResNet152、GRU 与 BiGRU) 以及是否使用预训练词向量。虽然并未完全控制变量，但是根据先前一次对 ResNet101 网络和 ResNet152 网络在表 4 的比较，两者所有召回指标加和分别为 341.0 和 341.1 十分接近，虽然不同任务下有些差别，但可以假设 ResNet101 和 ResNet152 网络在验证预训练词向量的实验中对准确率影响不大。

表 3: 使用预训练词向量对准确率是否有提升的实验

Model	Caption Retrieval					Image Retrieval				
	R@1	R@5	R@10	Med r	Mean r	R@1	R@5	R@10	Med r	Mean r
ResNet152+GRU	36.9	67.2	77.1	3.0	17.0	29.5	59.5	71.0	3.0	23.0
ResNet101+BiGRU	39.2	68.0	78.7	3.0	16.0	30.2	59.7	71.7	3.0	21.0
ResNet152+BiGRU$_{word2Vec}$	41.8	70.2	80.7	2.0	13.0	32.5	63.6	74.7	3.0	17.0

表 4: ResNet152 网络与 ResNet101 网络的比较

Model	Caption Retrieval					Image Retrieval				
	R@1	R@5	R@10	Med r	Mean r	R@1	R@5	R@10	Med r	Mean r
ResNet101	38.7	66.8	77.9	2.0	16.0	29.3	58.1	70.3	4.0	24.0
ResNet152	36.9	67.2	77.1	3.0	17.0	29.5	59.5	71.0	3.0	23.0

总的来看，可以认为使用预训练的词向量可以对模型准确率有进一步的提升，在很多实验中也都是默认使用词向量的。

3.5 使用不同类型模型实现 VSE

这里主要是指使用 GRU 越 CNN 结合的 VSE、使用 Bert[2] 与 ViT 结合的 VSE 以及使用 Bert 与 CNN 结合的 VSe 三种模型框架。其中 Bert 使用的 huggingface 上的bert-base-uncased 模型，ViT 使用的 huggingface 上的ViT-small，各模型架构下的最佳训练结果如表 5 所示。

表 5: 不同类型模型架构最佳训练结果

Model	Caption Retrieval					Image Retrieval				
	R@1	R@5	R@10	Med r	Mean r	R@1	R@5	R@10	Med r	Mean r
ResNet152+BiGRU	43.5	72.4	82.1	2.0	14.0	35.3	66.6	77.3	3.0	16.0
ViT+Bert	40.3	69.7	80.7	2.0	13.0	29.0	59.6	71.7	4.0	20.0
ResNet152+Bert	41.2	69.9	79.2	2.0	13.0	30.5	61.8	72.2	3.0	24.0

其中 ResNet152+BiGRU 模型使用 InfoNCE 损失，对 BiGRU 部分做平均池化，使用了 GoogleNews300 词向量，且微调了 ResNet152。ViT+Bert 模型使用了 InfoNCE 损失，ResNet152+Bert 模型也使用了 InfoNCE 损失函数。实验之前原本以为 ViT+Bert 这组会有较优的效果，但是并没有达到理想的效果。从原理上来看，或许是使用平均池化的操作让 BiGRU 对句子信息的提取更充分，以及 BiGRU 本身较少的参数量让其在与 ResNet152 模型相互对齐的过程中更容易吧。

3.6 在训练中途切换使用 max_violation

一开始我只在上一次任务中一份从 github 上 clone 下来基本没怎么修改的代码上成功使用过 max_violation。本次在我自己重构的代码中直接从 0 开始使用 max_violation 训练模型全都会陷入局部最优，都会达到 Loss 为 2*margin，即训练出的正样本和最困难的负样本完全抵消，导致梯度消失。后续翻阅了 github 上 vsepp 仓库的一些 issues，从issues#27中得到灵感，尝试先不使用 max_violation，而是在普通的链式损失上训练，达到 10 个 epoch 后再使用 max_violation 进一步训练。结果如表 6 中的 max_violation 所示。

表 6: 使用 max_violation 的效果

Model	Caption Retrieval					Image Retrieval				
	R@1	R@5	R@10	Med r	Mean r	R@1	R@5	R@10	Med r	Mean r
max_violation	42.3	69.0	80.1	2.0	12.0	29.8	59.7	71.7	3.0	21.0
InfoNCE	41.8	70.2	80.7	2.0	13.0	32.5	63.6	74.7	3.0	17.0

其中两者均使用 ResNet152 与 BiGRU 模型，均使用 GoogleNews300 词向量，同时也都对 BiGRU 取平均池化。但使用 max_violation 训练的模型虽然效果显著，但与使用 InfoNCE 的损失函数的模型还是有较小差距。

4 训练参数

训练设备为 4 张 2080 显卡，显存为 10.75G 左右。学习率取 0.0002，并在 15 个 epoch 后变为 0.00002。batch_size 设置为 128，对齐层的 embed_size 设置为 1024，词嵌入维度设置为 300，GRU 以及多头自注意机制默认层数为 2 和 6(手动传入)，margin 设置为 0.2，防止梯度爆炸的 grad_clip 设置为 2.0，默认训练 30 个 epoch，每 10 个 batch 在终端记录日志，每 500 个 batch 使用验证集验证一次。InfoNCE 损失中的

[2]其实也打算试一下使用 T5 模型，用其生成的第一个 token 作为对该句子的抽取试试的，但是一开始代码写错了，requires_grad 这个参数写成了 require_grad，导致虽然运行了但是没锁住 T5，我这边的服务器显卡跑不了，也就尝试用 bert 替代，后面改正这个错误后因为时间原因也并未再次尝试用 T5

temperature 默认设置为 0.1，但是加入参数更新。多头注意力机制 head 数分别使用过 3 和 6 两个，默认使用 3。

5 总结与心得

本次任务中的一个发现就是在很多次的训练中，总是会出现过拟合的现象，即使有对学习率的调整，在训练后期还是能从验证集上看到十分明显的准确率下降。此外就是本次学习到了使用 nohup 将程序托管在服务器上，并将终端输出保存为 log 文件。在本次任务中还有一些尝试，如给两个模态的 Encoder 后多加一些线性层，辅助对齐，但是结果呈现出来并不是很好，反而将效果往下拉了一点。我这次也认识到最好还是提前把需要跑的对比实验设计完全，否则最后到当时记录的数据中去找就总会发现一些实验没做等问题。