使用 paddle 复现 NGM 算法

姚瑾 519030910343

January 2022

1 算法介绍

图匹配 (graph matching) 是人工智能中的一个经典问题, 在若干领域都有重要的应用. 比如计算机视觉 (computer vision) 中匹配 2D/3D 形状, 生物信息学 (bioinformatics) 中匹配蛋白质网络, 社交网络中匹配不同网络当中的用户等。

图匹配的定义为给定两张拓扑图 (tological graph) G_1, G_2, 找到这两张图之间顶点的对应关系。

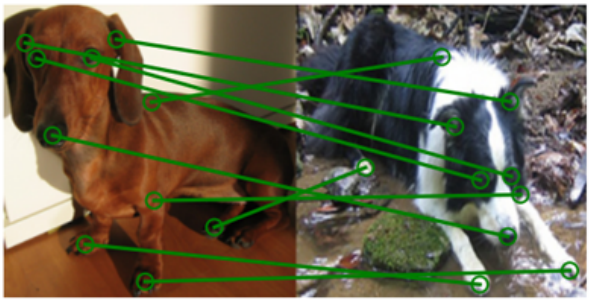

图 1: An example of graph matching

图匹配的结果一般由一个指派矩阵 (assignment matrix)X 表示，其中指派矩阵的每行、每列有且仅有一个元素为 1。我们采用一个同时包含一阶和二阶相似度信息的相似度矩阵 (affinity matrix)K 来建模图结构之间的相似度，它的对角线元素包含了节点与节点的相似度信息，非对角线元素包含了边与边的相似度信息。基于相似度矩阵 K 与指派矩阵 X，二图匹

配问题可以被公式化为 Lawler 形式的二次指派问题 (Lawler’s Quadratic Assignment Problem, Lawler’s QAP)，其公式如下：

$$\begin{aligned} & J(\mathbf{X}) = \mathrm{vec}(\mathbf{X})^\top \mathbf{K} \mathrm{vec}(\mathbf{X}) \\ s.t. \quad & \mathbf{X} \in \{0,1\}^{n_1 \times n_2}, \mathbf{X}\mathbf{1}_{n_2} = \mathbf{1}_{n_1}, \mathbf{X}^\top \mathbf{1}_{n_1} \leq \mathbf{1}_{n_2} \end{aligned} \tag{1}$$

NGM 算法的主要思路便是采用深度学习方法来求解 Lawler’s QAP，它的算法流程如下图所示：

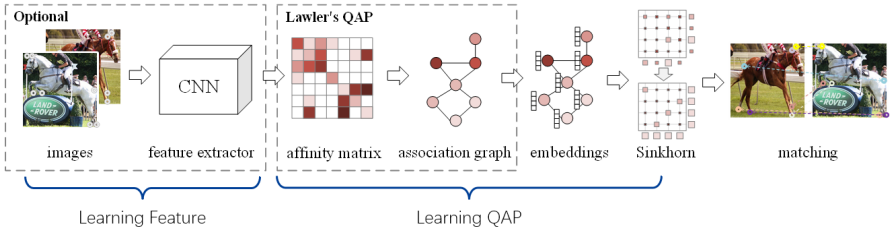

图 2: the pipeline of NGM method

CNN 网络层用于提取图片的边和点的特征，在具体实现中用 vgg16 网络来实现其功能。相似度学习层用来学习节点与节点之间，边与边之间的相似度，并且生成一个伴随图。通过伴随图的形式，图匹配问题转化为在伴随图上的节点分类问题。编码层将伴随图作为输入，使用图嵌入方法来学习伴随图的节点分类，将深度学习直接用于求解二次指派问题。sinkhorn 层将图嵌入得到的节点分数矩阵转化为一个双随机矩阵，最后使用 cross entropy 损失函数来对 sinkhorn 层得到的结果进行求导，并且梯度回传来更新网络参数。

注意到整个网络是端对端的学习方法，因此可以对整个流程进行联合调优。并且特征学习模块和 QAP 求解模块是可以分开的，即 NGM 方法超越了视觉领域的图像匹配问题，它所能处理的是最通用的二次指派问题（比如 Lawler’s QAP），其创新之处在于将相似度矩阵建模成伴随图，因此可以将 QAP 问题的求解转化为伴随图上的节点分类问题，因而可以采用深度学习方法来进行图嵌入。

2 复现 torch 版本

第一个困难是配置环境的问题，在我刚开始上手这次大作业时，还没有发放服务器的账号，我打算自己在自己电脑上安装环境，在了解了一些需要的库时，发现在 windows 系统上安装这些库比较麻烦，而且容易出一些问题，所以我在自己电脑上安装了 ubuntu 双系统，并且为了使用 ThinkMatch 官方给出的 docker 镜像，我自己学习了 docker 的相关知识，学会了使用 docker，并且在自己电脑上可以进行训练和测试的复现。这一工作在前期所占用的时间比较多。不过在助教提供了服务器后，配置环境可以直接使用 singularity 来构建容器，所以简单了很多。

在服务器上搭建好环境后，下载 ThinkMatch 官方提供的预训练模型，在主目录下新建了一个 pretrained 文件夹，用来存放下载的预训练模型，然后更改对应的模型参数配置文件，将”PRETRAINED_PATH” 更改为对应的位置，并且注意 GPUS 的设置，之后便可以测试 torch 版本的代码了。表 1 和表 2 分别为 torch 版本的 NGM 算法在 Pascal VOC 和 willow Object 数据集上的复现结果：

由于 Pascal VOC 数据集有 20 个类，表格较长，不便展示，这里只展示最后的 5 个类以及平均值，完整的测试结果和日志会在提交的文件中。

表 1: the result on Pascal VOC dataset (only the final 5 classes)

	pottedplant	sheep	sofa	train	tvmonitor	mean
precision	0.8300	0.6731	0.6230	0.8031	0.9000	0.6575
recall	0.8300	0.6731	0.6230	0.8031	0.9000	0.6575
f1	0.8300	0.6731	0.6230	0.8031	0.9000	0.6575
coverage	0.2057	0.2944	0.9643	0.6768	0.2442	
norm objscore	1.000866	1.00045	1.001106	1.000435	1.000741	1.001515
time	0.066999	0.076655	0.07859	0.089204	0.0628	0.076286

通过对比复现结果和论文给出结果，发现最终结果都是略好于论文给出的结果，这应该得益于官方给出的预训练模型性能比较好，可能是开源库中的预训练模型相比论文中使用的模型进行了参数调优。

表 2: the result on Willow Object Class dataset

	Car	Duck	Face	Motorbike	Winebottle	mean
precision	0.9731	0.8240	0.9808	0.8779	0.8096	0.8931
recall	0.9731	0.8240	0.9808	0.8779	0.8096	0.8931
f1	0.9731	0.8240	0.9808	0.8779	0.8096	0.8931
coverage	0.5474	0.2391	0.0272	0.5474	0.1005	
norm objscore	1.00028	1.000345	1.000006	1.000371	1.000992	1.000399
time	0.083264	0.081409	0.081574	0.096777	0.096417	0.087888

3 算法代码迁移

NGM 算法的主文件为 model 文件,在实现过程中还用到了 gnn, sinkhorn, build_graph, feature_align, factorize_graph_matching, geo_edge_feature, hungarian, evaluation_metric, 以及 GMN 中的 affinity_layer 文件，下面将详细说明这些文件的代码迁移。

表 3 是在代码迁移中我自己总结的 torch 和 paddle 的部分函数转换，在复现过程中翻阅 torch 和 paddle 的文档，比较不同函数的功能花费了较多时间，尤其是一些 torch 中支持的功能而 paddle 中不支持的功能，需要自己去实现。

表 3: 复现过程中总结的 torch 与 paddle 的函数转换表

torch	paddle
torch.cat(tensors, dim=0)	paddle.concat(x, axis=0)
torch.mul()	paddle.multiply()
F.normalize(..., dim = 2)	F.normalize(..., axis = 2)
torch.bmm	paddle.matmul
torch.Tensor.transpose(1,2)	paddle.Tensor.transpose((...2,1,...))
torch.Tensor.permute(0,2,1)	paddle.Tensor.transpose((0,2,1))
torch.Tensor.expand_as(x)	paddle.Tensor.expand_as(x)
torch.isnan(Tensor)	paddle.isnan(Tensor)
torch.mm	paddle.mm
torch.symeig	paddle.linalg.eigh
torch.nn.LocalResponseNorm	paddle.nn.LocalResponseNorm

Table 3 continued from previous page

torch	paddle
torch.cumsum(dim)	paddle.cumsum(axis)
Tensor.view	paddle.reshape
torch.repeat_interleave	paddle.to_tensor(numpy.repeat())

gnn 文件是我第一个进行代码迁移的文件，该文件实现了 NGM 算法要用到的图神经网络层。在该文件中，主要难点在于 torch 版本的前向计算使用了 2 种模式，一种是密集矩阵模式，另外一种是稀疏矩阵模式，由于 paddle 不支持稀疏矩阵的类，在尝试自己实现 paddle 稀疏矩阵类失败以后，我将 torch 的稀疏矩阵类都转换为了密集矩阵。其他需要注意的地方有：线性层在 paddle 中需要加入 bias_attr=True 的参数，否则得到的网络层结构不同。torch 中支持 repeat_interleave 函数，其作用为重复 Tensor 中的指定元素，在 paddle 中需要手动实现，这里采用了 paddle.to_tensor(numpy.repeat()) 来组合实现。其他的一些需要替换的函数已经在表 3 中列出。

sinkhorn 文件定义了 NGM 算法所用到的 sinkhorn 模块，它将输入的矩阵交替进行行和列的归一化，在多次执行后得到一个双随机矩阵。该文件代码迁移的困难在于设备的转换，这里用到了 place2str 函数，将设备转换为字符串形式再指定给数据。

在 factorize_graph_matching 文件中，和上面同样的原因，由于 paddle 不支持稀疏矩阵的运算，所以去掉了稀疏矩阵计算的优化。在 convert_params 文件中仿造已经实现的 pca 算法的参数转换编写了 ngm 算法的模型参数转换代码。其他文件的难点和上面的难点相同，便不再赘述。

4 paddle 复现结果

使用 NGM 算法在在 Pascal VOC 和 willow Object 数据集上测试二图匹配的结果分别如表 4 和表 5。由于 Pascal VOC 数据集有 20 个类，表格较长，不便展示，这里只展示最后的 5 个类以及平均值，完整的测试结果和日志会在提交的文件中。

表 4: the paddle reproduced result on Pascal VOC dataset (only the final 5 classes)

	pottedplant	sheep	sofa	train	tvmonitor	mean
precision	0.8509	0.6572	0.5944	0.7725	0.7456	0.6312
recall	0.8509	0.6572	0.5944	0.7725	0.7456	0.6312
f1	0.8509	0.6572	0.5944	0.7725	0.7456	0.6312
coverage	0.2061	0.2957	0.9643	0.6768	0.2412	
norm objscore	1.0002	1.0002	1.0004	1.0002	1.0011	1.0007
time	0.12099	0.1532	0.13873	0.09343	0.09993	0.17481

表 5: the paddle reproduced result on Willow Object Class dataset

	Car	Duck	Face	Motorbike	Winebottle	mean
precision	0.9689	0.8039	0.9943	0.8732	0.8381	0.8957
recall	0.9689	0.8039	0.9943	0.8732	0.8381	0.8957
f1	0.9689	0.8039	0.9943	0.8732	0.8381	0.8957
coverage	1.0000	1.0000	0.2612	1.0000	0.9662	
norm objscore	1.0001	1.0003	1.0000	1.0002	1.0006	1.0003
time	0.40244	0.39301	0.351	0.35627	0.3613	0.36498

5 结果分析

对于 Pascal VOC 数据集，使用 Pytorch 实现的 NGM 图匹配算法平均准确率为 0.6575，平均用时为 0.076286 s，使用 PaddlePaddle 框架实现的 NGM 图匹配算法平均准确率为 0.6312，平均用时为 0.17481 s.

对于 Willow Object Class 数据集，使用 Pytorch 实现的 NGM 图匹配算法平均准确率为 0.8931，平均用时为 0.087888 s，使用 PaddlePaddle 框架实现的 NGM 图匹配算法平均准确率为 0.8957，平均用时为 0.36498 s.

由以上数据可以看出，使用两个框架得到的最终平均准确率基本相同，但是平均测试时间相差较大，PaddlePaddle 版本比 Pytorch 版本慢 2-4 倍。分析其原因，主要在于 Pytorch 版本使用了稀疏矩阵来优化计算，使得计算速度大幅提升，而 PaddlePaddle 版本不支持稀疏矩阵，所以速度比较慢。

6 其他探索

除了 NGM 算法，我还对 NGM 系列的其他算法进行了代码迁移的尝试，即 NGM 的超图匹配，NGM 的多图匹配，以及 NGMv2 版本的二图、超图和多图匹配。这些算法的模型迁移代码已经基本写好，但是目前还无法完成预训练模型参数转换，并且由于时间原因没来得及进行适配，相关的代码会提交在附件中。

7 总结

在本次大作业中，完成了要求的工作：选择进行实验的图匹配开源算法，仔细阅读论文并深入理解算法。从 ThinkMatch 下载代码并且配置好 Pytorch 环境，复现论文的测试结果。参考 PaddlePaddle，将 Pytorch 格式的预训练模型转换为 PaddlePaddle 格式的预训练模型。将算法转换为 PaddlePaddle 框架下的可执行算法并且在数据集上验证精度。

通过本次大作业学习了图匹配的深度学习方法，对 PaddlePaddle 深度学习框架的使用更加熟练，提升了自己的工程能力和实践能力。