算法设计与分析大作业

22009200601 汤栋文

2024 年 12 月 12 日

目录

Code

https://github.com/MTDoven/Machine-Learning/tree/main/algorithm_design/permutation

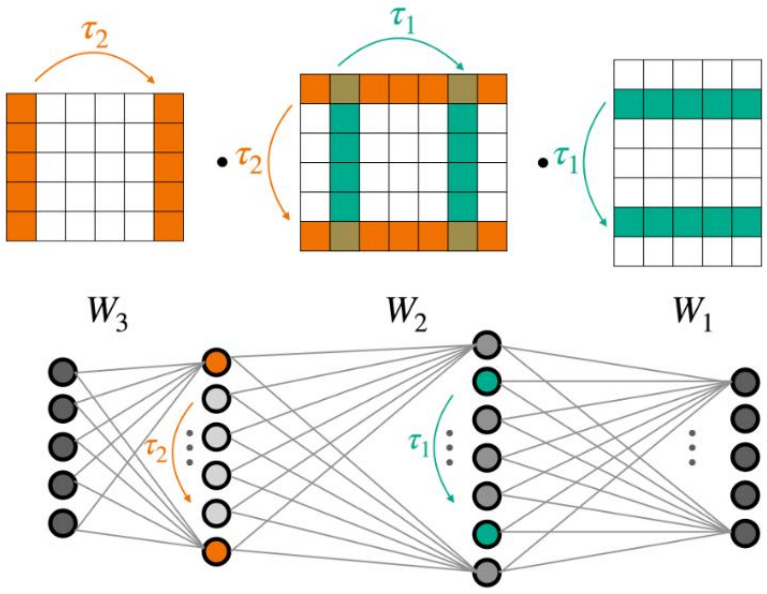
摘要

本研究探讨了神经网络权重空间中的对称性问题，分析了如何通过置换权重矩阵的行和列而不改变网络所表示的函数。我尝试使用匈牙利算法与集束搜索，将神经网络的对称状态进行对齐。具体而言，我将神经网络对称状态的调整建模为二部图完全匹配的基本模型。利用匈牙利算法被用来解决单层内的最佳匹配问题，而集束搜索则用于在多层间寻找全局较优解，平衡了解的质量和计算效率。除此之外我还进一步讨论了，神经网络的对称性除了置换对称性之外，还存在缩放不变性等其他形式的对称性。此外，我分享了关于 MWORKS 平台的体验，指出其作为科学计算工具的优点以及局限性，并提出了一些建议以提高其通用性和用户接受度。

1. 相关工作

1.1 神经网络对称性

在深度权重空间中的对称性，以一个三层的多层感知机为例（如下图所示）。对于任意逐点非线性激活函数σ，可以将置换 τ_1、τ_2 应用于 MLP 连续权重矩阵的行和列上，而不会改变该网络所表示的函数[2]。



具体来说，这意味着你可以交换神经网络中某一层的神经元顺序（即置换权重矩阵的行），同时相应地交换下一层神经元的连接权重（即置换权重矩阵的列），整个网络的输入-输出映射不会发生改变。这种特性是由于神经网络内部的对称性导致的，它允许在不改变模型功能的情况下重新排列权重。

1.2 神经网络参数理解与生成

HyperNetworks[3]提出了一种小型网络，用于为更大的网络生成各种架构的参数。2024 年，P-diff[4]等工作开始利用扩散过程来生成神经网络参数。此后 Cond P-diff[5]引入了任务或文本控制的参数生成方法。但这些所有的研究都受到一个问题的困扰，神经网络的对称性导致了相同功能的模型有成千上万种组合，这导致了学习过程极为困难。

ARCHITECTURE	NUM. PERMUTATION SYMMETRIES
MLP (3 layers, 512 width)	$10 \wedge 3498$
VGG16	$10 \wedge 35160$
ResNet50	$10 \wedge 55109$
Atoms in the observable universe	$10 \wedge 82$

2. 任务介绍

2.1 理论基础

为了简化问题，我们仅考虑顺序执行的全连接网络，其中W表示模型参数，l表示层数。我们对相邻两层的参数进行如下变换，不会改变最终的模型输出，其中P表示多个经过第二类行变换（交互任意两行）的初等矩阵的乘积。

$$\boldsymbol{z}_{\ell+1} = \boldsymbol{P}^\top \boldsymbol{P}\boldsymbol{z}_{\ell+1} = \boldsymbol{P}^\top \boldsymbol{P}\sigma(\boldsymbol{W}_\ell \boldsymbol{z}_\ell + \boldsymbol{b}_\ell) = \boldsymbol{P}^\top \sigma(\boldsymbol{P}\boldsymbol{W}_\ell \boldsymbol{z}_\ell + \boldsymbol{P}\boldsymbol{b}_\ell)$$

$$\boldsymbol{W}'_\ell = \boldsymbol{P}\boldsymbol{W}_\ell, \quad \boldsymbol{b}'_\ell = \boldsymbol{P}\boldsymbol{b}_\ell, \quad \boldsymbol{W}'_{\ell+1} = \boldsymbol{W}_{\ell+1}\boldsymbol{P}^\top,$$

2.2 建模与抽象

我们希望在一个l层的网络中，找到$l-1$个恰当的变换，使得源模型与目标模型之间参数的$MSELoss$最小。想要遍历所有的变换显然是不现实的，对于一个两层的全连接，如果隐层神经元数量为n那么总共的对称状态将有中A_n^n种，如果要全部遍历将会是阶乘复杂度。因此我们把问题分解为使单层$MSELoss$最小和使所有层$MSELoss$最小。

层内：我们首先考虑第一层的情况，由于输入神经元的排列顺序是确定的，因此们不需要考虑输入的情况（表现在参数矩阵上就是列的排序已经确定），因此我们只需要考虑输出的情况，既我们只需要考虑行的排序。也就是说我们需要找到 A 矩阵行的最佳排序使得 A 矩阵与 B 最为接近（这里定义为$MSELoss$最小）。再换言之，我们需要使 A 矩阵的行与 B 矩阵的行实现一个最佳匹配，到此为止第一层参数的变换建模成为了一个二部图匹配问题，因此我们使用二部图匹配的经典算法匈牙利算法来完成。

层间：根据上面层内的描述，我们可以从第一层递推，由于第一次的输出（行变换）已经确定，因此第二层的输入（列变换也将确定），所以我们可以用同样的方法确定第二层的行变换，直到倒数第二层，而倒数第一次由于输出的排序是确定的，而输入的排序是由倒数第二层确定的，所以最后一层不需要计算。但是显然这样做法不一定能找到全局最优解，为了让解尽可能让人满意，我考虑在单层时除了获取最优解以外再找到几个次优解，通过一个集束搜索（beam search）来平衡计算时间和解的满意程度。

3. 讲解算法

3.1 匈牙利算法

我们具体的匹配问题其实是一个完美匹配问题（即匹配覆盖了所有的顶点），通常也把这类问题叫做任务分配问题，因为这在任务分配中行常见。我们输入一个系数矩阵代表每对元素匹配的花销，我们希望让最终匹配的花销越小越好。具体来说该算法会涉及到以下步骤：

输入： 一个花销系数矩阵。

步骤一： 对系数矩阵执行变换。从系数矩阵的每行元素减去该行的最小元素，每列减去该列的最小元素，得到各行各列都有 0 元素的系数矩阵。

步骤二： 执行指派，找出独立的 0 元素。每行每列仅有一个独立 0 元素，其他的 0 元素为非独立 0 元素。如果独立零元素与矩阵的阶数相同则已经找到了最优解，则跳转到输出。

步骤三： 进行行列标记。1.对没有非独立 0 元素的行打√；2.对已打勾的行中含有 0 元素的列打√；3.对所有打勾的列中含有独立 0 元素的行打√；4.重复 2/3 直到打不出新的√为止；4.对没有打√号的行画一条横线，有打√号的列画一条纵线，这就得到覆盖所有 0 元素的最少直线。

步骤四： 对系数矩阵执行变换。在没有被直线覆盖的部分中找出最小元素。然后在打√行各元素中都减去这个最小元素，而在打√列的各元素上都加之这个最小元素。(保证原来 0 元素不变。) 如此得到新系数矩阵，而且它的最优解和原系数矩阵相同。返回步骤二。

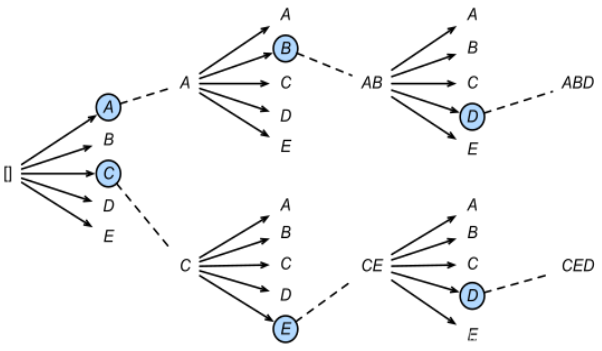
输出： 独立零元素的位置既为最优匹配。

这个算法说白了就是最大流的算法，但是它跟据二分图匹配这个问题的特点，把最大流算法做了简化，提高了效率，“对系数矩阵执行变换”这一步骤本质上就是在找最大流，但是它不需要建网络模型，所以图中不再需要源点和汇点，仅仅是一个二分图，每条边也不需要有方向。匈牙利算法能够在$O(n^3)$的时间复杂度内找到最优解。

3.2 集束搜索

对于花销系数矩阵是整数的情况下，输入不同排序的花销系数矩阵，匈牙利算法可能给出不同的匹配，但其实他们的都是最大匹配。因此我对花销系数矩阵离散化（离散化另一个重要原因是整数运算要远快于浮点运算），并且使用多个随机的排序，使匈牙利算法给出不同的输出。并对这些不同的输出进行集束搜索（beam search）。

Beam Search 通过设定一个固定的宽度（称为“beam width”），然后进行宽度优先搜索，在每个步骤中只保留最有可能的（花销最小的）若干个候选节点，而不是像宽度优先搜索那样保留所有可能的节点。这些被选中的节点会扩展到下一步，而其他未被选中的节点则被丢弃，这个过程一直持续到找到目标状态或者达到预定的最大深度。这种做法可以通过调节 beam width 使得解的满意程度和时间空间成本进行平衡。



4. 实验结果

4.1 实验结果

匈牙利算法虽然能得到最优解但是在计算时间上不尽人意，$O(n^3)$的复杂度对神经网络来说还是太高。这里仅采用一个很小的三层全连接在 MNIST 下的分类任务进行试验。

Beam width	*Original* $\boldsymbol{MSELoss}$	*Permutated* $\boldsymbol{MSELoss}$	*Time (s)*
1	10.56	9.68	11
10	10.56	7.39	102
50	10.56	6.54	445
100	10.56	6.47	876

根据实验结果可以发现，集束搜索功不可没，在不启用 beam search 时（即 beam width 为 1 时），输出参数的损失与原始相比较并没有明显的下降，而当 beam width 逐渐增大时，输出结果明显变好，并且在 50 附近稳定。但另一方面，时间开销也大幅度增加，因此无限增大 beam width 也并不可取。

4.2 进一步讨论

对于我的这个任务有一定的特殊性，它是在调整神经网络参数的对称性，它比较一般的轮换对称更加复杂，例如把前一层的参数整体乘以 10 再把后一层参数整体除以 10，在某些激活函数下输出的结果也不变，这也可以被看作是神经网络参数的对称性质之一。因此也许存在一些人类难以观察到的先验规律，一些基于神经网络推理过程的优化方法[1]也许更加实用，实际上经过调研我们后续的参数生成工作也打算使用此类方法进行数据的预处理，而不是使用传统的算法进行对齐。

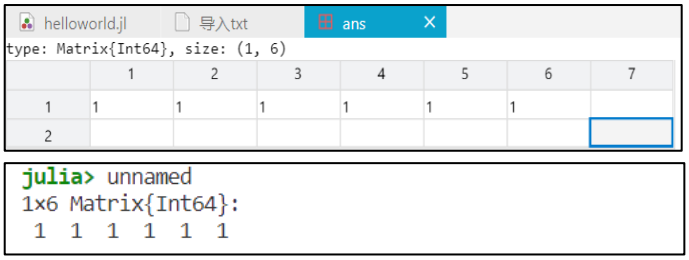
5. MWORKS 平台分析

5.1 整体感受

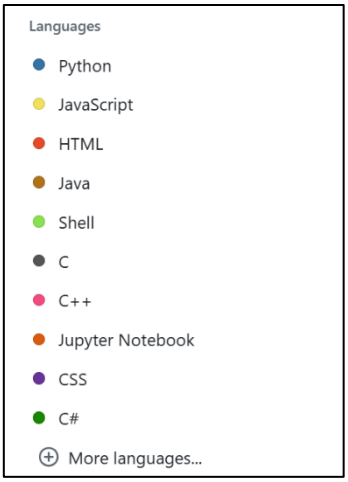
事先声明本人没用过 Matlab，做算法建模或可视化等工作全部使用 Python+Jupyterlab，虽说这款软件是对标 Matlab，但我不打算把它与 Matlab 进行对比。这款软件整体感觉很好上手，界面也简单清晰容易理解，感觉很适合用来验证或者学习一些简单算法，也包括一些编程语言的学习。作为刚起步的国内自主开发的科学计算软件来说是非常好用了。

5.2 优点与不足

优点：如果去掉上方的任务栏，你会发现这款软件的操作模式和界面与 VSCode 几乎完全一致，一个熟悉的界面布局与操作让这款软件更容易上手。可视化界面导入数据和查看数据的功能对于做数据分和预处理来说非常方便，支持多种常见的数据文件格式，不需要用写代码来加载数据文件。



不足：让我比较纳闷的一点是，为什么选择 Julia 为基本语言？实话说这门的确非常简单，但它并不通用，熟悉 Python 的人可以在十几分钟内搞清楚这门语言的基本使用方法。但为什么不直接用 Python+Numpy 呢，我猜想这次作业真正用 Julia 写的人并不会太多，可能大多数人还是把它当成一个写 Python 的 IDE 来使用。虽然学习成本很低但也需要一点学习成本的，而且学到的东西几乎仅在这个软件上可用，这极大的限制了这个软件的推广。



Github 上我需要点开 More languages 手动搜索才能找到 Julia 语言

5.3 其他建议

将数据处理的相关功能，包括“数据绘图”，“表格可视化”等功能与 Numpy 和 Matplotlib 等工具结合。最好能够实现“可视化地使用 Matplotlib”，这既能兼容绝大多数人先前的工具与代码，又提供了足够的便利。这个软件现在的模式感觉做出来了一个介于 SPSS 和 VSCode 之间的东西，这当然是没问题的，但需要提高通用性或者解决大多数人的痛点才会有更多人使用，否则只会让人觉得这只是某某软件不太行的平替。

6. 结论

本研究深入探讨了神经网络中权重空间的对称性问题，并针对由此引发的参数生成难题提出了一种解决方案。通过将匈牙利算法应用于单层内部的最佳匹配问题，以及利用集束搜索（beam search）在多层间寻找全局较优解。实验结果表明，该方法能够在保证模型功能不变的前提下，显著减少源模型与目标模型之间的差距。然而好的效果同时伴随着计算成本的上升，因此在实际应用中需要根据具体需求权衡解的质量和计算效率，这也提示我们可以进一步探索基于神经网络推理过程的优化方法，以实现更加高效的参数预处理和生成。此外，我分享了关于 MWORKS 平台的体验，指出其作为科学计算工具的优点以及局限性，并提出了一些建议。

引用

[1] Git Re-Basin: Merging Models modulo Permutation Symmetries
[2] Equivariant Architectures for Learning in Deep Weight Spaces
[3] HyperNetworks
[4] Neural Network Parameter Diffusion
[5] Conditional LoRA Parameter Generation
[6] https://www.cnblogs.com/nickchen121/p/15499576.html
[7] https://www.zhiguf.com/focusnews_detail/1425159
[8] https://blog.csdn.net/qq_52302919/article/details/132170356