深度学习调参手册

这不是一个官方支持的谷歌产品。

Varun Godbole[†], George E. Dahl[†], Justin Gilmer[†], Christopher J. Shallue[‡], Zachary Nado[†]

† Google Research, Brain Team

‡ Harvard University

Table of Contents

这份文件为谁准备?

这份文件是为深度学习模型性能极为感兴趣的工程师和研究人员（包括个人和团队）准备的。我们假设其有一定机器学习和深度学习概念及基本知识。

我们的重点是超参数调整过程。同时也触及了深度学习训练的其他方面，如管道实现和优化，但对这些方面的讲述并不完整。

我们假设机器学习问题是一个监督学习问题或看起来很像一个问题的东西（如自监督）。也就是说，本文中的一些建议可能适用于其他类型的问题。

为什么需要调参手册?

目前，要想让深度神经网络在实践中运行良好，需要付出惊人的努力和猜想。更糟糕的是，人们很少记录获得良好结果的设置。论文掩盖了导致其最终结果的过程，以呈现一个更干净的故事，而从事商业问题的机器学习工程师很少有时间概括他们的调参过程。教科书倾向于回避实践指导，优先考虑基本原则，即使其作者有必要应用工作经验来提供有效的建议。在准备创建这份文件时，我们找不到任何全面的说明来实际解释如何用深度学习获得好的结果。相反，我们在博客文章和社交媒体上发现了一些建议的片段，从研究论文的附录中窥见了一些技巧，偶尔有关于某个特定项目或管道的案例研究，还有很多困惑。 深度学习专家和不太熟练的从业者使用表面上类似的方法，但取得的结果间存在着巨大的差距。同时，这些专家也欣然承认他们所做的一些事情可能没有充分的理由。随着深度学习的成熟并对世界产生更大影响，社区需要更多资源，涵盖有用的设置，包括所有的实际细节，这对获得良好的结果至关重要。

我们是由5名研究人员和工程师组成的团队，他们已经在深度学习领域工作了很多年，其中一些人早在2006年就开始工作了。我们已经将深度学习应用于语音识别、天文学等各种问题，并在这一过程中学到了很多。这份文件是从我们自己训练神经网络、教导新的机器学习工程师以及为我们的同事提供深度学习实践建议的经验中发展出来的。 尽管看到深度学习从少数学术实验室实践的机器学习方法变成了为数十亿人使用的产品提供动力已经令人欣慰，但作为一门工程学科，深度学习仍然处于起步阶段，我们希望这份文件能鼓励其他人帮助该领域实现实验协议系统化。

这份文件是我们试图清理我们自己的深度学习方法时产生的，因此它代表了作者在写作时的观点，而不是任何形式的客观真理。我们自己在超参数调整方面的挣扎使其成为我们指导的一个重点，但我们也涵盖了在工作中遇到的其他重要问题（或看到的错误）。我们的意图是让这项工作成为一份活的文件，随着我们信仰的改变而不断成长和发展。例如，关于调试和减轻训练失败的材料，我们两年前是不能写的，因为它是基于最近的结果和正在进行的调查。不可避免地，我们的一些建议将需要更新，以考虑到新的结果和改进的工作流程。我们不知道最佳的深度学习配方，但在社区开始写下和辩论不同的过程之前，我们无法找到它。 为此，我们鼓励发现我们的建议有问题的读者提出代替意见，并提供令人信服的证据，以便我们能够更新手册。我们也希望看大可能有不同建议的指南和手册，以便我们能够作为一个社区努力实现最佳过程。 最后，任何标有😈 的部分都是我们想做更多研究的地方。只有在尝试写完这本手册后，才完全明白在深度学习从业者的工作流程中可以找到多少有趣的和被忽视的研究问题

新项目指南

我们在调参过程中做出的许多决定可以在项目开始时做一次，只有在情况发生变化时才会偶尔重新审核。

我们的指导意见做出以下假设：

- 问题制定、数据清理等基本工作已经完成，在模型架构和训练配置上花时间是有意义的。
- 已经有一个管道设置好了，可以做训练和评估，而且很容易为各种感兴趣的模型执行训练和预测工作
- 适当的衡量标准已经被选择和实施。这些指标应尽可能地代表在部署环境中所测量的内容。

选择模型架构

概述： *当开始一个新项目时，尽量重用一个已经工作的模型。*

- 选择一个成熟的、常用的模型架构，先让它工作起来。以后可以建立一个自定义模型
- 模型架构通常有各种超参数，决定模型的大小和其他细节（如层数、层宽、激活函数的类型）
    - 因此，选择模型架构实际上意味着选择一个不同的模型系列（每个模型超参数的设置都只有一个）
    - 我们将在选择初始配置和提高模型性能的科学方法中考虑选择模型超参数的问题。

- 在可能的情况下，尽量找到一篇解决手头问题的论文，并以该模型为起点进行复制。

选择优化器

概述： *从当前问题中最流行的优化器开始*

- 在所有类型的机器学习问题和模型架构中，没有哪个优化器是"最好"的。即使只是comparing the performance of optimizers is a difficult task.😈
- 我们建议使用成熟的、受欢迎的优化器，特别是在开始一个新项目时。
  - 理想情况下，选择用于同类型问题最流行的优化器
- 对所选优化器的**所有**超参数给予关注。
  - 具有更多超参数的优化器可能需要更多的调整努力来找到最佳配置。
  - 这在项目的开始阶段尤其重要，因为我们正试图找到其他各种超参数（如架构超参数）的最佳值，同时将优化器超参数视为nuisance parameters.
  - 在项目的最初阶段，可能最好从一个比较简单的优化器开始(如带动量的SGD或者Adam带 ϵ, β_1和
    β_2) 以后再换成一个更通用的优化器
- 我们喜欢的成熟的优化器包括（但不限于）：
  - SGD with momentum(我们更喜欢Nesterov变体)
  - Adam and NAdam，比带动量的SGD更普遍。请注意，Adam有4个可调整的超参数，他们可能都是重要的and they can all matter!
    - 参考
      How should Adam's hyperparameters be tuned?

选择batch size

概述： *batch size控制着训练速度，不应该被用来直接调整验证集的性能。通常情况下，理想的batch size是可用硬件所支持的最大batch size*

- batch size是决定训练时间和计算资源消耗的一个关键因素
- 增加batch size往往会减少训练时间。这可能是非常有益的，因为：
  - 可以在一个固定的时间内对超参数进行更彻底的调整，可能会产生一个更好的最终模型
  - 减少开发周期，可以更频繁的测试新想法
- 增加batch size可能会减少、增加或不改变资源消耗
- batch size不应该被当作验证集性能的课调整超参数
  - 只要所有的超参数都调的很好（尤其是学习率和正则化超参数），并且训练step的数量足够多，使用任何batch size都应该可以达到相同的最终性能（参考Shallue et al. 2018)
  - 参考 Why shouldn't the batch size be tuned to directly improve validation set performance?

确定可行的batch size并估计训练工作量

▶ *[Click to expand]*

- 对于一个给定的模型和优化器，通常会有一个可用硬件支持的batch size范围。限制性因素通常是加速器内存。
- 不幸的是，如果不允许或至少不编译完整的训练程序，就很难计算出多大的batch size适合于内存。

- 最简单的解决方案通常是以不同的批量大小（例如增加2的幂）运行训练程序，进行少量的steps，直到其中一个程序超过了可用的内存。
- 对于每个batch size，我们应该训练足够长的时间，以获得对训练吞吐量的可靠估计。
- 当加速器还没有饱和时，如果批处理量增加一倍，训练吞吐量也应该增加一倍（或者至少接近一倍）。等价地，随着批处理量的增加，每步的时间应该是恒定的（或者至少是接近恒定的）。
- 如果不是这样，那么训练管道就有一个瓶颈，如I/O或计算节点之间的同步。这可能值得在继续进行之前进行诊断和纠正。
- 如果训练吞吐量只增加到某个最大batch size，那么我们应该只考虑到该最大吞吐量，即使硬件支持更大的batch size。
    - 使用较大的batch size的所有好处都是假设训练的吞吐量增加。如果没有，那就解决瓶颈问题或使用较小的batch size。
    - **梯度累积（Gradient accumulation）**模拟的batch size比硬件所能支持的要大，因此不能提供任何吞吐量的好处，一般来说，在应用工作中应避免使用它。
- 这些steps可能需要在每次改变模型或优化器时重复进行（例如，不同的模型架构可能允许更大的批处理量以适应内存）

选择batch size尽可能减少训练时间

▶ *[Click to expand]*

$$\text{训练时间} = (\text{每steps时间}) \times (\text{总steps})$$

- 我们通常可以认为当没有并行计算开销，并且所有的训练瓶颈都已被诊断和纠正时，每一步的时间对于所有可行的batch size来说都是近似恒定的。（参考上一节for how to identify training bottlenecks ）在实践中，通常至少有一些增加batch size大小的开销。
- 随着batch size的增加，达到固定性能目标所需的总steps通常会减少（前提是当batch size大小改变时，所有相关的超参数都被重新调整，参考Shallue et al. 2018）。
    - 例如，增加一倍的批处理量可能使所需的总步骤数减半。这就是所谓的 **完美缩放。**
    - 完美的缩放性对所有的batch size都是成立的，直到一个临界batch size，超过这个临界batch size，就会取得递减的回报
    - 最终，增加batch size不再减少训练steps（但也不会增加）。
- 因此，使训练时间最小化的batch size通常是仍能减少所需训练steps的最大batch size。
    - 这个batch size取决于数据集、模型和优化器，除了对每一个新问题进行实验发现之外，如何计算它是一个问题。 
    - 在比较batch size时，要注意示例预算/epoch预算（在固定训练例子展示数量的情况下运行所有试验）和steps预算（在固定训练steps数量的情况下运行所有试验）之间的区别
        - 用一个历时预算来比较batch size，只能探测到完美的缩放尺度，即使更大的batch size可能仍然通过减少所需的训练steps来提供加速。
    - 通常情况下，可用硬件支持的最大batch size将小于临界batch size。因此，一个好的经验法则（没有进行任何实验）是使用尽可能大的批处理量。
- 如果最终增加了训练时间，那么使用更大的batch size是没有意义的。

选择batch size以尽量减少资源消耗

▶ *[Click to expand]*

- 有两种类型的资源成本与增加batch size有关：
  1. *前期成本*，如购买新硬件或重写训练管道以实现多GPU/多TPU训练
  2. *使用成本*，例如，根据团队的资源预算计费，从云供应商处计费，电力/维护成本
- 如果增加batch size有很大的前期成本，那么推迟增加batch size可能更好，直到项目成熟，更容易评估成本效益的权衡。实施多主机并行训练可能会引入错误和微小问题，因此，无论如何，一开始就用一个比较简单的管道可能会更好。（另一方面，当需要进行大量的调整实验时，训练时间的大幅加速可能会在早期非常有利）
- 我们把总的使用成本（可能包括多种不同类型的成本）称为"资源消耗"。我们可以将资源消耗分解为以下几个部分。

$$\text{资源消耗} = (\text{setp资源消耗}) \times (\text{总steps})$$

- 增加batch size通常可以减少总steps。资源消耗是增加还是减少，将取决于每个step的消耗如何变化。
  - 增加batch size可能会减少资源消耗。例如，如果大batch size每个step都可以在与小batch size相同的硬件上运行（每一步只增加少量时间），那么每个step资源消耗的增加可能被step数的减少所抵消。
  - 增加batch size可能不会改变资源消耗。例如，如果将batch size增加一倍，所需steps减少一半，所使用的GPU数量增加一倍，总消耗量（以GPU小时计）将不会改变。
  - 增加batch size可能会增加资源消耗。例如，如果增加batch size需要升级硬件，每step消耗的增加可能超过steps的减少

改变batch size需要重新调整大多数超参数

▶ *[Click to expand]*

- 大多数超参数的最优值都对batch size很敏感。因此，改变batch size通常需要重新开始调整过程
- 与batch size相互作用最强的超参数是优化器超参数（如学习率、动量）和正则化超参数，因此对每个batch size进行单独调整最为重要
- 在项目开始时选择批量大小时要记住这一点。如果你以后需要切换到不同的批量大小，为新的批量大小重新调整一切，可能会很困难，浪费时间，也很昂贵。

batch norm如何与batch size相互作用

▶ *[Click to expand]*

- batch norm很复杂，一般来说，应该使用与梯度计算不同的batch size来计算统计数据。详细讨论参考batch norm部分

选择初始配置

- 在开始超参数调整之前，我们必须确定起点。这包括指定（1）模型配置（如层数），（2）优化器超参数（如学习率），（3）训练steps数量
- 确定初始配置将需要一些手动配置以及训练运行和试错
- 我们的指导原则是找到一个简单、相对快速、相对低资源消耗的配置，获得"合理"的结果

    - “简单”意味着尽可能的避免使用“华丽的搭配”，这些可以在以后添加。即使这些”华丽的搭配“被证明是有帮助的，在最初的配置中加入它们也有可能浪费时间来调整无用的功能和/或加入不必要的复杂因素。
        - 例如在增加花哨的衰减时间表之前，先从一个恒定的学习率开始
    - 选择一个快速的、消耗最少资源的初始配置，将使超参数调节的效率大大提升。
        - 例如，从一个较小的模型开始。
    - "合理的"性能取决于问题，但至少意味着训练好的模型在验证集上的表现比随机选择好得多（尽管它可能差到不值得部署）。
- 选择训练steps的数量涉及平衡以下紧张关系：
    - 一方面， 更多steps的训练可以提高性能，并使超参数的调整更容易 (参考Shallue et al. 2018).
    - 另一方面，较少steps的训练意味着每次训练运行的速度更快，使用的资源更少，通过减少周期之间的时间来提高调参效率，并允许更多的实验被并行运行。此外，如果最初选择了一个不必要的大steps预算，那么接下来可能就很难改变了，例如，一旦学习率计划被调整为该步数。

提高模型性能的科学方法

就本文而言，机器学习开发的最终目标是最大化所部署模型的效用。即使开发过程的许多方面在不同的应用中有所不同（如时间长度、可用的计算资源、模型的类型），我们通常可以在任何问题上使用相同的基本步骤和原则。

我们下面的指导意见做出了以下假设：

- 已经有一个完全运行的训练管道，以及一个获得合理结果的配置。
- 有足够的计算资源可用于进行有意义的调参实验，并至少并行运行几个训练作业。

渐进式调整策略

概述： *从简单的配置开始，逐步进行改进，同时建立对问题的洞察力。确保任何改进都是基于强有力的证据，以避免增加不必要的复杂性。*

- 我们的最终目标是找到一种配置，使我们的模型性能最大化。
    - 在某些情况下，我们的目标将是在一个固定的截止日期前最大限度地提高模型的性能（例如提交给一个比赛）。
    - 在其他情况下，我们希望无限期地改进模型（例如，不断地改进生产中使用的模型）。
- 原则上，我们可以通过使用一种算法来自动搜索整个可能的配置空间，从而最大限度地提高性能，但这并不是一个实用的选择。
    - 可能的配置空间非常大，目前还没有任何足够成熟的算法可以在没有人类指导的情况下有效地搜索这个空间。
- 大多数自动搜索算法依赖于手工设计的搜索空间，该空间定义了要搜索的配置集，而这些搜索空间可能相当重要。
- 将性能最大化的最有效方法是从简单的配置开始，逐步增加功能并进行改进，同时建立对问题的洞察力。
    - 我们在每一轮调整中使用自动搜索算法，并随着我们理解的加深而不断地更新我们的搜索空间。
- 随着我们的探索，我们自然会发现越来越好的配置，因此我们的 "最佳 "模型将不断改进。
    - 当我们更新我们的最佳配置时，我们称之为启动(launch)（可能是也可能不是对应于一个生产模型的实际启动）。

    - 对于每一次启动，我们必须确保这种变化是基于强有力的证据--而不是基于幸运的配置的随机机会--这样我们就不会给训练管道增加不必要的复杂性。

在高层次上，我们的增量调整策略包括重复以下四个步骤：

1. 为下一轮的实验确定一个适当范围的目标。
2. 设计并运行一套实验，使之在这一目标上取得进展。
3. 从结果中学习
4. 考虑是否推出新的最佳配置。

本节的其余部分将更详细地探讨这一策略。

探索与开发

概述： *大多数时候，我们的首要目标是深入了解问题。*

- 尽管人们可能会认为我们会花大部分时间试图在验证集上实现性能最大化，但在实践中，我们花了大部分时间试图深入了解问题，而相对来说很少有时间贪婪地关注验证损失。
    - 换句话说，我们把大部分时间花在 "探索 "上，只有少量的时间花在 "开发 "上。
- 从长远来看，如果我们想最大限度地提高我们的最终绩效，了解问题是至关重要的。将洞察力放在优先于短期收益的位置可以帮助我们：
    - 避免启动不必要的变化，这些变化恰好存在于表现良好的运行中，只是由于历史上的偶然。
    - 识别验证损失对哪些超参数最敏感；哪些超参数相互影响最大，因此需要一起重新调整；哪些超参数对其他变化相对不敏感，可以在实验中固定下来。
    - 如果出现了过拟合问题，建议尝试潜在的新特征，如新的正则器。
    - 识别那些没有帮助的特征，可以将其删除，减少实验的复杂性。
    - 识别超参数调整带来的改进何时可能达到饱和。
    - 围绕最优值缩小我们的搜索空间，以提高调参效率。
- 当我们最终准备好的时候，可以纯粹地关注验证误差，即使实验对调参问题的结构没有太多信息。

为下一轮实验选择目标

概述： *每一轮的实验都应该有一个明确的目标，而且范围要足够窄，以便实验能够真正朝着目标取得进展。*

- 每一轮实验都应该有一个明确的目标，而且范围要足够窄，使实验能够真正朝着目标前进：如果我们试图一次增加多个功能或回答多个问题，我们可能无法区分对结果的单独影响。
- 目标示例包括：
    - 尝试对管道进行潜在的改进（例如，新的正则化方式、预处理选择等）。
    - 了解特定模型超参数（如激活函数）的影响
    - 最大程度的减少验证集损失

设计下一轮实验

概述： *确定指标超参数、需要调整超参数和固定超参数，以实现实验目标。建立一个研究序列来比较指标超参数的不同值，同时对调整超参数进行优化。选择调整超参数的搜索空间，以平衡资源成本和科学价值。*

识别指标、调整和固定超参数

▶ *[Click to expand]*

- 对于一个给定的目标，超参数分为指标、调整和固定
  - 指标超参数是指用于评估参数对模型效果的参数
  - 调整超参数是值那些需要优化的参数，以便公平地比较指标超参数的不同值。这与统计学中的调整参数概念类似。nuisance parameters.
  - 固定超参数在本轮实验中不变。这些超参数的值在比较指标超参数时不需要（或我们不希望它们）改变。
    - 通过为一组实验固定某些超参数，我们必须接受从实验中得出的结论可能对固定超参数的其他设置无效。换句话说，固定的超参数为我们从实验中得出的任何结论创造了限制条件。
- 例如，如果我们的目标是 "确定具有更多隐藏层的模型是否会减少验证误差"，那么隐藏层的数量就是一个指标超参数。
  - 学习率是一个麻烦的超参数，因为我们只有在为每个层数分别调整学习率的情况下，才能公平地比较具有不同层数的模型（最佳学习率一般取决于模型结构）。
  - 如果我们在之前的实验中确定激活函数的最佳选择对模型深度不敏感，或者我们愿意限制隐藏层数量，只观察激活函数的具体选择，那么激活函数可以是一个固定的超参数。另外，如果我们准备为每一个隐藏层的数量分别进行调整，它也可以是一个调整参数。
- 一个特定的超参数是指标超参数、调整超参数还是固定超参数，并不是固定的，而是根据实验目标而变化。
  - 例如，激活函数的选择可以是一个指标超参数（ReLU或tanh是更好的选择吗？），一个调整超参数（当我们允许几个不同的激活函数时，最好的5层模型是否比最好的6层模型更好？），或一个固定的超参数（对于ReLU激活函数，在特定位置添加batch normalization是否有帮助？）
- 在设计新一轮的实验时，我们首先为实验目标确定指标超参数。
  - 在这个阶段，我们认为所有其他超参数都是调整超参数。
- 接下来，我们把一些调整超参数转换成固定的超参数。
  - 在资源无限的情况下，我们会把所有非指标超参数作为调整超参数，这样我们从实验中得出的结论就不会有固定超参数值的限制了。
  - 然而，我们试图调整的调整超参数越多，就越有可能无法在每一个指标超参数的设置上充分调整，最终从实验中得出错误的结论。
    - 如下所述,我们可以通过增加计算预算来应对这一风险，但往往我们的最大资源预算少于在所有非科学的超参数上进行调整所需的资源。
  - 当我们判断固定超参数引入比把它作为一个调整超参数的成本低时，我们会选择把一个调整超参数转换成一个固定的超参数。
    - 一个给定的调整超参数与指标超参数的相互作用越大，固定其数值的破坏性就越大。例如，权重衰减强度的最佳值通常取决于模型的大小，因此，假设一个单一的权重衰减特定值来比较不同的模型大小，不会有很好的说服力。
- 尽管我们给每个超参数分配的类型取决于实验目标，但对于某些类别的超参数，我们有以下经验法则。
  - 在各种优化器的超参数中（如学习率、动量、学习率计划参数、Adam betas等），至少有一些会成为调整超参数，因为它们往往与其他变化互动最多。
    - 它们很少是指标超参数，因为像 "当前管道的最佳学习率是多少？"这样的目标并不能提供太多的说服力--无论如何，最佳设置很容易随着下一个管道的改变而改变。

        - 由于资源的限制，或者当我们有特别有力的证据证明它们与指标参数没有相互作用时，我们可能会偶尔固定其中的一些参数。但一般来说，我们应该假定优化器的超参数必须单独调整，以便在科学超参数的不同设置之间进行公平的比较，因此不应该被固定。
            - 此外，我们没有先验的理由去选择一个优化器的超参数值而不是另一个（例如，它们通常不会以任何方式影响正向传递或梯度的计算成本）。
    - 相比之下，优化器的选择通常是指标超参数或固定超参数。
        - 如果我们的实验目标涉及在两个或多个不同的优化器之间进行公平的比较（例如 "确定哪个优化器在给定的步骤数中产生最低的验证误差"），它就是一个指标超参数。
        - 另外，我们可能出于各种原因将其作为一个固定的超参数，包括：（1）先前的实验使我们相信，对我们的问题来说，最好的优化器对当前指标超参数不敏感；（2）我们更愿意使用这个优化器来比较科学超参数的值，因为它的训练曲线更容易推理；（3）我们更愿意使用这个优化器，因为它比替代方案使用的内存少。
    - 正则化技术引入的超参数通常是调整超参数，但我们是否包含正则化技术则是一个指标或固定超参数。
        - 例如，dropout会增加代码的复杂性，因此在决定是否包括它时，我们会将是否添加dropout作为一个指标超参数，而将dropout rate作为一个调整超参数。
            - 如果我们决定在这个实验的基础上将dropout加入管道，那么在未来的实验中，dropout rate将是一个调整超参数。
    - 架构超参数通常是指标或固定超参数，因为架构的变化会影响服务和训练成本、延迟和内存需求
        - 例如，层数通常是一个指标或固定超参数，因为它往往会对训练速度和内存使用产生巨大的影响。
- 在某些情况下，调整和固定超参数的集合取决于指标超参数的值。
    - 例如，假设我们试图确定在Nesterov momentum和Adam中哪一个优化器能使验证误差最小。 指标超参数是 `optimizer`，其取值为 `{"Nesterov_momentum", "Adam"}`。当值为 `optimizer="Nesterov_momentum"` 包含调整/固定超参数 `{learning_rate, momentum}` 当值为 `optimizer="Adam"` 包含调整/固定超参数 `{learning_rate, beta1, beta2, epsilon}`.
    - 只有在指标超参数的某些数值下才会出现的超参数被称为 **条件超参数**。
    - 我们不应该仅仅因为两个条件超参数有相同的名字就认为它们是相同的！在上面的例子中，条件超参数名为 `learning_rate` 是不同的超参数当 `optimizer="Nesterov_momentum"` 和 `optimizer="Adam"`。它在两种算法中的作用相似（虽然不完全相同），但在每种优化器中效果良好的数值范围通常有几个数量级的差异。

创建一套研究报告

▶ *[Click to expand]*

- 一旦我们确定了指标和调整超参数，我们就会设计一个 "研究 "或一系列研究，以便朝着实验目标取得进展。
    - 一项研究指定了一组为后续分析而运行的超参数配置。每个配置被称为一个 "试验"。
    - 创建一个研究通常包括选择不同试验中变化的超参数，选择这些超参数的取值（"搜索空间"），选择试验的数量，并选择一个自动搜索算法来从搜索空间中抽取该数量的试验。另外，我们也可以通过手动指定超参数配置的集合来创建一个研究。
- 研究的目的是用不同的科学超参数值运行管道，同时 "优化掉"（或 "优化"）调整超参数，以便指标超参数不同值之间的比较尽可能公平。
- 在最简单的情况下，我们会对指标参数的每个配置进行单独研究，对调整超参数进行调整。

    - 例如，我们的目标是在Nesterov momentum和Adam中选择最佳优化器，，我们可以创建一个研究，在其中`optimizer="Nesterov_momentum"`调整参数为`{learning_rate, momentum}`，另一项研究中`optimizer="Adam"`调整参数为`{learning_rate,beta1, beta2, epsilon}`。 我们将通过从每个研究中选择表现最好的试验来比较两个优化器。
    - 我们可以使用任何无梯度的优化算法，包括贝叶斯优化或进化算法，对调整超参数进行优化，尽管我们更倾向于在探索阶段使用准随机搜索进行调整，因为它在这种情况下有多种优势。在探索结束后，如果有最先进的贝叶斯优化库，这就是我们的首选。
- 在更复杂的情况下，我们想比较大量的指标超参数，而做那么多独立的研究是不现实的，我们可以把指标参数和调整超参数列入同一个搜索空间，并使用搜索算法在一个研究中对指标和调整超参数进行抽样。
    - 当采取这种方法时，条件超参数会引起问题，因为除非调整超参数集对指标超参数的所有值都是相同的，否则很难指定一个搜索空间。
    - 在这种情况下，我们更倾向于使用准随机搜索而不是更复杂的黑箱优化工具，因为它能确保我们获得相对统一的指标超参数值的抽样。不管是哪种搜索算法，我们都需要以某种方式确保它能均匀地搜索科学参数。

在内容丰富的实验和负担得起的实验之间取得平衡

▶ *[Click to expand]*

- 在设计一项研究或一系列研究时，我们需要分配有限的预算，以充分实现以下三个愿望：
    1. 比较足够多的指标超参数
    2. 在足够大的搜索空间内调整调整超参数。
    3. 对调整超参数的搜索空间进行足够密集的采样。
- 我们越能实现这三个愿望，我们的实验越有说服力
    - 尽可能多地比较指标超参数的值，扩大我们从实验中获得说服力的范围。
    - 包括尽可能多的调整超参数，并允许每个调整超参数在尽可能大的范围内变化，这增加了我们的信心，即对于指标超参数的每个配置，在搜索空间**存在**一个 "好 "的调整超参数值。
        - 否则，我们可能会在指标超参数的值之间进行不公平的比较，因为我们没有搜索调整参数空间的可能区域，在那里指标参数可能存在更好的值。
    - 尽可能密集地对调整超参数的搜索空间进行采样，增加我们的信心，即任何恰好存在于我们搜索空间中的良好调整超参数都会被搜索程序发现。
        - 否则，我们可能会在指标参数的值之间进行不公平的比较，因为有些值在调整超参数的抽样中变得更幸运。
- 不幸的是，这三个维度中的任何一个改进都需要增加试验的数量，从而增加资源成本，或者找到一种方法来节省其他维度的资源。
    - 每个问题都有自己的特异性和计算限制，所以如何在这三个方面分配资源需要一定程度的领域知识。
    - 在运行一项研究后，我们总是试图了解该研究是否对调整超参数进行了足够好的调整（即对足够大的空间进行了足够广泛的搜索），以公平地比较科学的超参数（如下文更详细地描述）。

从实验结果中提取灵感

概述： *除了努力实现每组实验的原始科学目标外，还要通过附加问题的检查表，如果发现问题，要修改实验并重新运行。*

- 最终，每组实验都有一个具体的目标，我们要评估实验为实现这一目标提供的证据。

    - 然而，如果我们提出正确的问题，我们往往会发现在一组特定的实验能够朝着最初的目标取得很大进展之前需要纠正的问题。
        - 如果我们不问这些问题，我们可能会得出不正确的结论。
    - 由于运行实验可能是昂贵的，我们也想借此机会从每组实验中提取其他有用的见解，即使这些见解与当前的目标没有直接关系。
- 在分析一组特定的实验，使其向最初的目标迈进之前，我们应该问自己以下的附加问题。
    - 搜索空间是否足够大？
        - 如果一项研究的最佳点在一个或多个维度上接近搜索空间的边界，那么搜索的范围可能不够大。在这种情况下，我们应该用扩大的搜索空间再进行一次研究。
    - 我们是否从搜索空间中抽出了足够的点？
        - 如果不是，就多跑点几次，或者在调整目标上不那么精益求精。
    - 在每项研究中，有多大比例的实验是 **不可行的** （即试验出现分歧，得到非常糟糕的损失值，或者因为违反了一些隐含的约束条件而根本无法运行）？
        - 当研究中很大一部分点是**不可行的**，我们应该尝试调整搜索空间，以避免对这些点进行采样，这有时需要重新参数化搜索空间。
        - 在某些情况下，大量的不可行点可能表明训练代码中存在错误。
    - 该模型是否表现出优化问题？
    - 我们能从最佳试验的训练曲线中学到什么？
        - 例如，最佳试验的训练曲线是否出现了过拟合现象。
- 如有必要，根据上述问题的答案，完善最近的研究（或一组研究），以改进搜索空间或对更多的试验进行抽样，采取一些其他纠正措施。
- 一旦我们回答了上述问题，我们就可以继续评估实验为实现我们最初的目标所提供的证据(例如，评估一个变化是否有用).

识别不良的搜索空间边界

▶ *[Click to expand]*

- 如果一个搜索空间中的最佳采样点接近其边界，那么这个搜索空间就是可疑的。如果我们向那个方向扩大搜索范围，我们可能会找到一个更好的点。
- 为了检查搜索空间的边界，我们喜欢将完成的试验绘制在**基本超参数轴图**上。我们将验证目标值与其中一个超参数（例如学习率）进行对比。图上的每一个点都对应于一个试验。
    - 每次试验的验证目标值通常应该是它在训练过程中取得的最佳值。

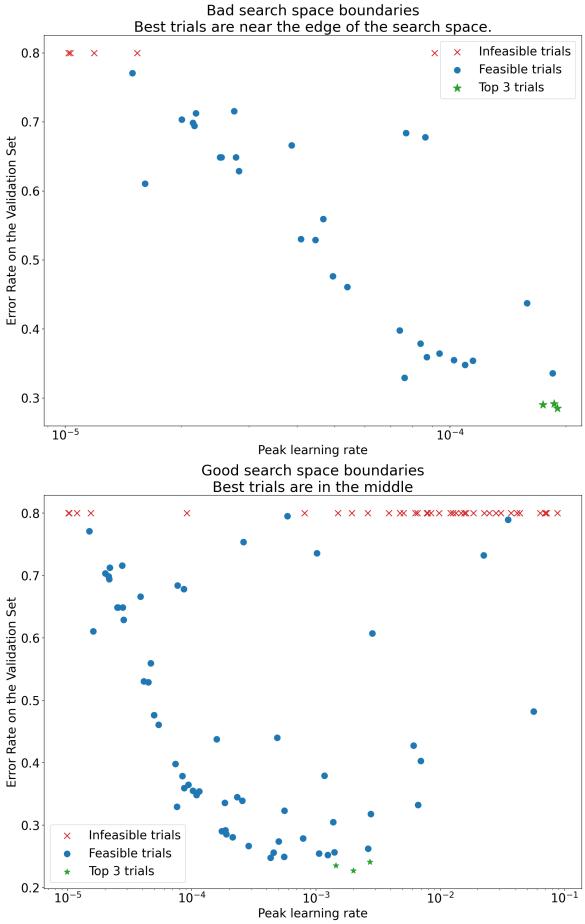

图 1:不良搜索空间边界和良好搜索空间边界例子

- 图 1 显示错误率（越低越好）与初始学习率的对比。
- 如果最佳点聚集在搜索空间的边缘（在某些维度上），那么搜索空间的边界可能需要扩大，直到最佳观察点不再靠近边界。
- 通常，一项研究会包括 "不可行 "的试验，这些试验会出现分歧或得到非常糟糕的结果（在上图中用红色X标记）。
    - 如果所有的试验对于学习率大于某个阈值的情况下是不可行的，并且如果表现最好的试验的学习率处于该区域的边缘。 该模型可能存在稳定性问题，使其无法获得更高的学习率。

在搜索空间中未对足够的点进行采样

▶ *[Click to expand]*

- 一般来说，要知道搜索空间的采样是否足够密集是非常困难的
- 进行更多的试验当然更好，但也有明显的代价。
- 由于很难知道我们什么时候采样够了，所以我们通常会对我们能负担得起的东西进行采样，并试图反复观察各种超参数轴图中校准我们的直觉，并试图了解有多少点在搜索空间的 "好 "区域。

检查训练曲线

▶ *[Click to expand]*

概述： *检查训练曲线是识别常见故障的一个简单方法，可以帮助我们确定下一步要采取行动的优先次序。*

- 虽然在很多情况下，我们实验的主要目标只需要考虑每一次试验的验证损失，但当把每一次试验简化为一个单一的数字时，我们必须小心，因为它可能会隐藏表面之下的重要细节。
- 对于每项研究，我们总是至少看最好的几项试验的**训练曲线**（训练损失和验证损失与训练steps在训练期间的关系图）。
- 即使这对解决主要的实验目标没有帮助，检查训练曲线也是识别常见故障模式的一个简单方法，可以帮助我们确定下一步要采取行动的优先次序。
- 在研究训练曲线时，我们对以下问题感兴趣。
- 是否有试验表现出**过拟合**现象？
    - 当验证误差在训练过程中的某个时刻开始增加时，就可能会出现过拟合现象。
    - 在实验环境中，我们通过为每个指标超参数设置选择 "最佳 "试验来优化调整超参数，我们应该在每个实验中检查**过拟合**问题。
        - 如果有一个最佳试验表现出过拟合现象，我们通常要用额外的正则化技术重新进行试验，或在比较指标超参数的值之前更好地调整现有的正则化参数。
            - 如果指标超参数包括正则化参数，这可能就不适用了，因为那样的话，正则化参数设置过低导致出现过拟合，也就不奇怪了。
        - 减少过拟合通常是直接使用常见的正则化技术，这些技术略微增加了代码复杂性或额外的计算（例如dropout、label smoothing、weight decay），在下一轮实验中增加一个或多个这样的技术通常没什么大不了的。
        - 例如，如果指标超参数是 "隐藏层的数量"，而使用最大数量隐藏层的最佳试验表现出过拟合现象，那么我们通常倾向于用额外的正则化来再次尝试，而不是立即选择较小数量的隐藏层。
        - 即使没有一个 "最好的 "试验表现出有问题的过度拟合，但如果在任何试验中出现这种情况，仍然可能有问题。
            - 选择最好的试验会抑制出现过拟合的配置，而偏向于那些没有问题的配置。换句话说，它将有利于具有更多规范化的配置。
            - 然而，任何使训练变得更糟糕的东西都可以作为正则器，即使它并不打算这样做。例如，选择一个较小的学习率可以通过阻碍优化过程来规范训练，但我们通常不希望以这种方式选择学习率。
            - 因此，我们必须意识到，为每一个指标超参数设置的 "最佳 "试验可能是某些指标或调整超参数产生了“坏”的结果。
- 在训练后期，训练或验证损失是否存在较高的step与step之间的差异？
    - 如果是这样，这可能会干扰我们比较不同指标超参数值的能力（因为每个试验都随机地在一个 "幸运 "或 "不幸运 "的步骤上结束），以及我们在生产中重现最佳试验结果的能力（因为生产模型可能不会在研究中的相同 "幸运 "步骤上结束）。
    - 最有可能导致步与步之间差异的原因是批次差异（从每批训练集中随机抽取例子），小的验证集，以及在训练后期使用过高的学习率。
    - 可能的补救措施包括增加批次大小，获得更多的验证数据，使用学习率衰减，或使用Polyak平均法。
- 训练结束后，试验是否仍在改进？
    - 如果是这样，这表明我们处于“计算约束”状态，我们可能会从增加训练steps的数量或改变学习率策略中得到改善。
- 在最后的训练步骤之前，训练集和验证集的性能是否早已饱和？
    - 如果是这样，这表明我们处于“不受计算约束”状态，我们可能会从增加训练steps的数量或改变学习率策略中得到改善。

- 虽然我们不能一一列举，但还有许多其他的行为可以通过检查训练曲线而变得明显（例如，训练损失在训练过程中*增加*，通常表明训练管道中存在错误）。

用隔离图检测参数变化是否有用

▶ *[Click to expand]*

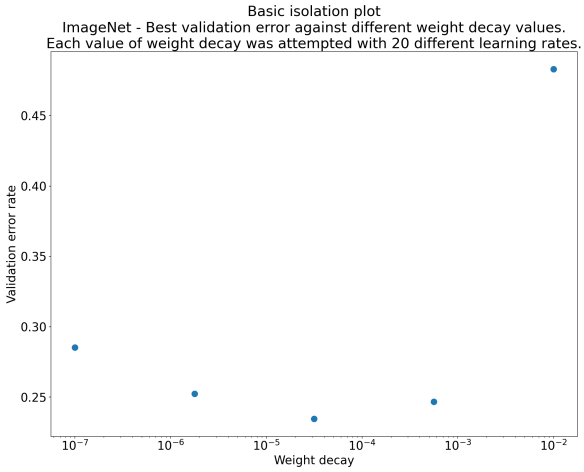

图 2: ImageNet上训练的ResNet-50最佳权重衰减值隔离图

- 通常，一组实验的目标是比较一个指标超参数的不同值。
    - 例如，我们可能想确定导致最佳验证损失的权重衰减值。
- **隔离图**是基本超参数轴图的一个特例。隔离图上的每一个点都对应于一些（或全部）调整超参数的最佳试验的性能。
    - 换句话说，我们在 "优化 "了调整超参数之后，绘制了模型的性能。
- 隔离图使其更容易在指标超参数的不同值之间进行对应的比较。
- 例如，图 2 显示了在ImageNet上训练的ResNet-50的特定配置下产生最佳验证性能的权重衰减值。
    - 如果我们的目标是确定是否要加入权重衰减，那么我们将把这个图中的最佳点与没有权重衰减的基线进行比较。为了进行公平的比较，基线也应该对其学习率进行同样良好的调整。
- 当我们有由（准）随机搜索产生的数据，并考虑用一个连续的超参数来绘制隔离图时，我们可以通过对基本超参数轴图的X轴值进行分桶，并在分桶定义的每个垂直切片中取最佳试验来近似绘制隔离图。

自动生成常用检查图谱

▶ *[Click to expand]*

- 绘制的图谱越多，我们就能从中获得更多信息。因此，我们有必要设置基础模块，以自动参数尽可能多的图谱
- 至少，我们会对实验中改变的所有超参数自动生成基本的超参数轴图。
- 此外，我们为所有的试验自动生成训练曲线，并尽可能方便地找到每项研究中最好的几个试验，检查它们的训练曲线。
- 我们还可以添加许多其他潜在的绘图和可视化的东西，这些都是有用的。虽然上面描述的那些是一个很好的起点，但套用Geoffrey Hinton的话说，"每当你绘制新的东西，你就会学到新的东西。

确定是否采用训练管道变化或超参数配置

概述： *当决定是否对我们的模型或训练程序进行改变或采用新的超参数配置前进时，我们需要意识到结果中不同的变化来源。*

- 当我们试图改进模型时，可能会观察到，与现有的配置相比，一个特定的候选变化最初取得了很好的验证损失，但在重复实验后发现，没有一致的优势。非正式地，我们可以把导致这种不一致结果最重要的变化来源分为以下几大类。
  - **训练程序方差**, **再训练方差**, 或 **试验方差**： 当我们使用相同的超参数，但不同的随机种子时会导致训练过程的差距。
    - 例如，不同的随机初始化、training data shuffles、dropout masks、数据增强操作模式和并行算术操作顺序，都是试验差异的潜在来源。
  - **超参数搜索方差**，或**学习方差**：由与选择搜索超参数的方式引起的结果差异。
    - 例如，我们可能用一个特定的搜索空间运行同一个实验，但用两个不同的种子进行准随机搜索，最后选择不同的超参数值。
  - **数据收集和抽样差异**：任何一种随机分成训练、验证和测试数据的差异，或更普遍地由于训练数据生成过程而产生的差异。
- 使用严格的统计测试对有限验证集上估计的验证错误率进行比较是很好的，但往往仅试验方差就能在使用相同超参数设置的两个不同的训练模型之间产生统计上的显著差异。
- 在试图做出超越超参数空间中单个点的水平结论时，我们最关心的是研究方差。
  - 研究方差取决于试验的数量和搜索空间，它比试验方差大以及小得多的情况都有可能发生。
- 因此，在采用一个候选变化之前，考虑运行最佳试验N次，以描述试验的差异。
  - 通常情况下，我们可以在管道发生重大变化后只对试验差异进行重新定性，但在某些应用中，我们可能需要更频繁的估计。
  - 在其他应用中，表征试验方差的成本太高，不值得。
- 在试验结束时，虽然我们只想采用能产生真正改进的变化（包括新的超参数配置），但要求完全确定某种东西有帮助，也不是完全。
- 因此，如果一个新的超参数点（或其他变化）得到了比基线更好的结果（尽可能考虑到新点和基线的再训练方差），那么我们可能应该把它作为新的基线，用于未来的比较。
  - 然而，我们应该只采用那些产生的改进超过其增加复杂性带来收益的。

探索结束后

概述： *一旦我们完成了对搜索空间的探索，并决定了哪些超参数应该被调整，贝叶斯优化工具就是一个引人注目的选择。*

- 在某些时候，我们优先考虑产生一个最佳配置，滞后调参问题。
- 在这一点上，应该有一个精炼的搜索空间，它舒适地包含了最佳观察试验周围的局部区域，并且已经被充分地采样了。
- 我们的探索工作应该已经揭示了最基本的超参数的调整（以及它们的合理范围），我们可以利用尽可能大的调整预算来构建一个搜索空间，以进行最终的自动调整。
- 由于我们不再关心参数带来的灵感，随机搜索许多优势不再适用，应该使用贝叶斯优化工具来自动寻找最佳超参数配置。
  - 如果搜索空间包含非显著的发散点（得到NaN训练损失或比平均值差很多的训练损失点），使用适当处理发散试验的黑匣优化工具是很重要的 (参考带未知约束的贝叶斯优化是处理这个问题一个很好的方法)。
- 在这一点上，我们还应该考虑检查测试集上的性能。

- 原则上，我们甚至可以将验证集折叠到训练集中，重新训练用贝叶斯优化法找到的最佳配置。然而，这只适合于未来不会有这种特定工作量的启动（例如，一次性的Kaggle比赛）。

确定训练steps

- 有两种类型的工作负载：一种是与计算有关的，另一种是与计算无关的。
- 当训练受**计算限制**时，训练受限于我们愿意等待多长时间，而不是受限于有多少训练数据或其他因素。
  - 在这种情况下，如果我们能以某种方式延长训练时间或提高训练效率，我们应该看到较低的训练损失，并且通过适当的调整，改善验证损失。
  - 换句话说，*加快*训练速度就等于改善训练，而 "最佳 "训练时间总是 "在我们能承受的范围内"。
  - 也就是说，工作负荷是有限的，并不意味着更长/更快的训练是提高结果的唯一途径。
- 当训练**不受计算限制**时，我们可以负担得起我们想训练多久就训练多久，而且，在某些时候，训练时间越长越没有帮助（甚至会导致过拟合问题）。
  - 在这种情况下，我们应该期望能够训练到非常低的训练损失，以至于训练时间越长可能会稍微减少训练损失，但不会很大幅度地减少验证损失。
  - 特别是当训练不受计算限制时，充足的训练时间预算可以使调谐更容易，特别是在调谐学习率衰减时间表时，因为它们与训练预算有特别强的互动。
    - 换句话说，匮乏的训练时间预算可能需要将学习率衰减计划调整到完美状态，以达到良好的损失。
- 不管一个给定的工作负载是否有计算限制，增加梯度方差（跨批次）的方法通常会导致较慢的训练进度，从而可能增加达到特定验证损失所需的训练步骤。高梯度方差可能是由以下原因造成的。
  - 使用小的batch size
  - 增加数据增量
  - 添加某些类型的正则化（如dropout）。

当训练不限时时应该训练多长时间

- 我们的主要目标是确保训练时间足够长，以使模型达到最好的结果，同时避免训练steps的数量过多。
- 在有问题的情况下，请选择较长的训练时间。假设回视（最佳）检查点选择得当，并且足够频繁，那么训练时间越长，性能就越不会下降。
- 不要在试验中调整`max_train_steps`的值。选择一个常数，并将其用于所有实验。从这些试验中，观察各训练steps中回视检查点，以完善`max_train_steps`的选择
  - 例如，如果最佳step总是在训练steps的前10%，那么`max_train_steps`就太高了。
  - 另外，如果最佳step一直在训练steps的后25%，更长的训练时间和重新调整的衰减策略可能会更进一步提升性能。
- 当架构或数据发生变化时，理想的训练steps数量会发生变化（例如增加数据增量）。
- 下面我们将描述如何使用恒定学习率确定合适的训练集所需steps，为`max_train_steps`挑选一个初始候选值。
  - 请注意，我们并不是以精确或数学上明确的方式使用 "完全适合训练集 "这一短语。它只是作为一个非正式的描述词，表示训练损失非常小。
    - 例如，当用log loss进行训练时，如果没有正则化，我们可能会看到训练损失一直在缓慢提高，直到达到浮点极限，因为网络权重无限制地增长，模型对训练集的预测变得越来越自信。在这种情况下，我们可以说，错误分类误差在训练集上达到零的时候，模型 "完全适合 "训练集。

- 如果训练过程中的梯度噪声增加，`max_train_steps`的起始值可能需要增加。
    - 例如，如果在模型中引入数据增强或正则器（如dropout）。
- 如果训练过程有一定的改善，可能会减少`max_train_steps`。
    - 例如，用一个更好的优化器或一个更好的学习率衰减策略。

使用学习率扫描为`max_train_steps`挑选初始值

▶ *[Click to expand]*

- 这个程序假定不仅有可能 "完美 "地拟合训练集，而且可以使用恒定的学习率计划来实现。
- 如果有可能完全适合整个训练集，那么一定存在一个配置（某个`max_train_steps`的值）完全适合训练集；找到这样的配置并使用其`max_train_steps`的值作为起点`N`。
- 在没有数据增强和正则化的情况下，运行恒定的学习率扫描（即网格搜索学习率），其中每个试验训练了`N`步。
- 我们对`max_train_steps`的初步猜测是，在扫描中最快达到完美训练性能所需的steps。
- **注意**：不良的搜索空间会具有欺骗性。
    - 例如，如果一项试验中的所有学习率都太小，我们可能会错误地得出结论，认为一个非常大的`max_train_steps`值是必要的。
    - 至少，我们应该检查试验中的最佳学习率是否处于搜索空间的边界。

当训练由计算机控制时应该训练多长时间

- 在某些情况下，训练损失一直在无限期地改善，我们的耐心和计算资源成为限制因素。
- 如果训练损失（甚至是验证损失）一直在无限期地改善，那么我们是否应该一直训练，只要我们能够负担得起？不一定。
    - 通过运行大量较短的实验，并为我们希望推出的模型保留最长的 "production length"，也许能够更有效地进行调整。
    - 随着试验的训练时间接近我们的耐心极限，调参实验对我们潜在的模型候选来说变得更加相关，但我们能完成的实验却更少。
    - 可能有很多问题我们可以在只训练~10%的训练长度时回答，但总是有一种风险，即我们在这个时间限制下的结论将不适用于20%的训练长度的实验，更不用说100%了。
- 在多轮训练中进行调参，并增加每轮训练的steps限制，是一种明智的做法。
    - 我们可以想做多少轮就做多少轮，但通常1-3轮是最实际的。
    - 从本质上讲，尽量使用周转时间非常快的试验来获得对问题的理解，用调整的彻底性和与最终最长运行的相关性来交换。
    - 一旦每一次试验的时间限制产生了有用的见解，我们就可以增加训练时间并继续调整，根据需要反复检查从较短的运行中得出的结论。
- 作为一个起点，我们建议进行两轮的调整：
    - Round 1：缩短运行时间，以找到好的模型和优化器超参数。
    - Round 2：在良好的超参数点上进行很少的长时间运行，以获得最终模型。
- 从`Round i` → `Round i+1`最大的问题是如何调整学习率衰减策略。
    - 在调整各轮学习率策略时，一个常见的陷阱是用太小的学习率来使用所有的额外训练步骤。

Round 1

▶ *[Click to expand]*

- 不幸的是，不能保证在短期不完整的训练中发现的好的超参数在训练长度大幅增加时仍然是好的选择。然而，对于某些种类的超参数来说，它们往往有足够的相关性，因此第一回合是有用的。
- 在较短的运行中发现的哪些超参数值，能转移到较长的训练运行中去？对于所有这些，我们需要更多的研究。但是根据我们目前所了解的情况，以下是作者的猜测，按照概率递减的顺序。
    - 非常有可能转移
        - 早期训练的不稳定性可以在第一轮调整中用较少的训练步骤来解决。也许这些超参数是我们所拥有的最接近于肯定的转移的东西。
            - Warmup length
            - Initialization
    - 有可能转移
        - 模型架构中的戏剧性胜利通常会转移，但可能有许多反例。
    - 可能会转移
        - 优化算法/优化器超参数 - 我们认为这将 "松散地 "转移。它肯定比上面的东西要弱。
        - Data augmentation
        - Regularization
            - 如果不可能完全拟合训练集，模型可能处于正则化不太可能有很大帮助的状态。
    - 不太可能转移
        - Learning rate schedule：不太可能完美转移。
            - 这篇文章表面即使学习率衰变策略可以转移，但我们不认为这在一般情况下是真的。比如。在小的训练step上调整sqrt衰减，然后扩展到大的步数，会导致大部分训练发生在过小的step上。
                - 在极端训练预算的限制下，人们可能会对大多数时间表做得 "足够好"，但如果对其进行调整，可能会看到明显的性能改进。
            - 理解随机元优化中的短跨度偏差 描述了试图短视地挑选学习率的危险性。

Round 2

▶ *[Click to expand]*

- 运行第一回合的最佳超参数配置。
- **(猜测)** 😈 利用额外的steps来延长高学习率的训练期。
    - 例如，如果是线性计划，那么从第一轮开始就保持衰减的长度不变，并在开始时延长恒定lr的时间。
    - 对于余弦衰减，只需保留第一轮的基础lr，并像Chinchilla论文中那样扩展 `max_train_steps`。
- 对于拥有非常成熟的建模和调参管道以及长时间的生产培训运行团队来说，更多的轮次可能是有意义的，但它们往往是矫枉过正。
    - 我们已经描述了如何从Step 1 → Step 2。如果我们不关心时间，有效利用计算是压倒一切的关注点，那么理想的做法是在许多不同的调整回合中以指数形式增加训练运行的长度（从而增加完成一项研究的端到端时间）
        - 在每一轮比赛中，我们系统地确保选择继续保持。

- 新的想法要经过一个管道，从Step i → Step i+1，使用越来越长的实验来逐步解读它们。

对训练管道的补充指导

优化输入管道

概述： *输入受限于管道的原因和干预措施在很大程度上取决于任务；使用剖析器并注意常见的问题。*

- 使用适当的分析器来诊断输入约束管道。比如JAX的Perfetto或TensorFlow的TensorFlow profiler。
- 最终，具体的原因和干预措施将高度依赖于任务。更广泛的工程考虑（如最大限度地减少磁盘占用）可能会导致输入管道性能变差。
- 常见的原因：
  - 数据没有与训练过程同步，导致I/O延迟（这可能发生在通过网络读取训练数据时）。
  - 昂贵的在线数据预处理（考虑离线做一次并保存）。
  - 非故意的同步障碍，干扰了数据管道读取。例如，在CommonLoopUtils中设备和主机之间同步度量时。
- 常见提示：
  - Instrument input pipeline to prefetch examples (例如tf.data.Dataset.prefetch)
  - 尽可能早地从每一个管道中删除未使用的功能/数据。
  - 增加为输入管道生成例子的复制。例如，通过使用tf.data service.

评估模型性能

概述： *以比训练更大的批次规模运行评估。以定期的步骤间隔，而不是定期的时间间隔来运行评估。*

评估设置

▶ *[Click to expand]*

- 我们可以在几种情况下评估我们模型的性能
  - **在线评估** - 当模型在生产环境中提供预测服务时，会收集指标。
  - **离线评估** - 当模型在生产环境中提供预测服务时，会收集指标。
  - **定期评估** - 在模型训练期间收集的指标可能是离线评估的代理，或在离线评估中使用的数据子集。
- 在线评估是黄金标准，但在模型开发阶段往往是不现实的。
- 根据问题的不同，离线评估可能是相当复杂的，而且计算成本很高。
- 定期评估是最实用和经济的选择，但可能不能完全代表生产环境。
  - 在定期评估期间，我们的目标是使用离线评估的便捷代理，而不牺牲我们在训练期间得到的信息可靠性。

设置定期评估

▶ *[Click to expand]*

- 我们在训练过程中进行定期评估，以实时监测其进展，便于回顾性地选择模型检查点，这样我们就可以在训练结束后检查训练曲线。
- 最简单的配置是在同一个计算实例中进行训练和定期评估，定期交替进行训练和评估。

    - 在这种情况下，用于执行评估的批次大小至少应该和用于训练的批次大小一样大，因为在评估过程中不需要维护模型的激活，降低了每个例子的计算要求。
- 定期评估应该以固定的steps间隔进行，而不是以时间间隔进行。
    - 基于时间间隔进行评估会使解释训练曲线变得更加困难，特别是当训练可能受到训练作业的抢占、网络延迟问题等的影响。
- 验证/测试指标的周期性（当使用洗牌的训练/验证/测试分割时）可以显示错误，如测试数据与训练数据重叠，或训练数据没有被正确洗牌。以定期的steps间隔进行评估可以使这些问题更容易被发现。
- 当评估集不能被batch size所分割时，就会出现部分批次。确保被填充的例子有正确的权重，以防止损失函数被它们所偏离。通常，这些被填充的例子可以被赋予零的权重。
- 每次评估要保存足够的信息以支持离线分析。理想情况下，我们会保存对个别例子的预测，因为它们对于调试是非常有价值的。
    - 生成类似SavedModels，可以在评估工作结束后轻松进行临时的模型检查。

选择定期评估的样本

▶ *[Click to expand]*

- 定期评估工作的运行速度可能不够快，无法在合理的时间内计算完整的离线评估集指标。这往往需要对定期评估的数据进行抽样。
- 在构建抽样数据集时，我们考虑以下因素。
    - 样本数量
        - 检查定期工作所使用的抽样数据集上计算的性能是否与整个离线评估集上的性能相匹配，即抽样集和完整数据集之间没有偏差。
        - 用于定期评估的数据集应该足够小，以便于在其整体上生成模型预测，但又不能太小，以便于准确测量模型的改进（即不被标签噪声所淹没）。
        - 要避免随着时间的推移对验证集进行适应性的 "拟合"，而这种拟合方式并不能推广到被保留的测试集。然而，这种考虑很少成为实际问题。
    - 不平衡的数据集
        - 对于不平衡的数据集，在罕见类别的例子上表现往往会有噪音。
        - 对于在一个类别标签中只有少量例子的数据集，对预测正确的例子数量进行记录，以便更深入地了解准确率的提高（0.05的灵敏度提高听起来令人振奋，但这只是多了一个正确的例子吗？）。

保存检查点并回视最佳检查点

概述： *运行训练的固定步数，并回顾性地从运行中选择最佳检查点。*

- 大多数深度学习框架支持模型检查点。也就是说，模型的当前状态会周期性地保存在磁盘上。这使得训练工作对计算实例的中断具有弹性。
- 最好的检查点往往不是最后一个检查点，特别是当验证集的性能并没有随着时间的推移而持续增加，而是围绕一个特定的值波动时。
- 设置管道，以跟踪训练期间迄今看到的N个最佳检查点。在训练结束时，模型的选择就是选择训练中看到的最佳检查点。我们把这称为**回顾性的最佳检查点选择**。
- 选用前瞻性的早停策略通常是没有必要的，因为我们预先规定了试验预算，并保留了迄今为止看到的N个最佳检查点。

设置实验跟踪

概述： *在跟踪不同的实验时，一定要注意一些要领，如研究中检查点的最佳表现，以及对研究的简短描述。*

- 我们发现，在电子表格中记录实验结果对我们所做的各种建模问题很有帮助。它通常有以下几栏。
  - 试验名称
  - 该试验对应的配置文件
  - 注释或对研究的简短描述
  - 运行的试验数量
  - 在研究中最佳检查点的验证集上的表现
  - 具体的复制命令或说明有哪些未提交的修改是启动培训所必需的
- 找到一个至少能捕捉上述信息的跟踪系统，并且对做实验的人来说是方便的。没有跟踪的实验还不如不存在。

Batch normalization操作细节

概述： *现在batch norm往往可以用LayerNorm代替，但在不能代替的情况下，在改变批量大小或主机数量时有一些棘手的细节。*

- batch norm使用当前批次的平均数和方差对激活进行归一化，但在多设备设置中，除非明确同步，否则这些统计数据在每个设备上都是不同的。
- 传闻（主要是在ImageNet上）说，batch size为64来计算这些归一化的统计数据在实践中效果更好。(参考
  Ghost Batch Norm this paper).
- 将总batch size和用于计算batch norm统计的例子数量解耦，对batch size的确定非常有用。
- Ghost batch norm并不总是能正确地处理每个设备的批大小>虚拟批大小的情况。在这种情况下，我们实际上需要对每个设备上的 batch norm进行再抽样，以便获得适当数量的 batch norm统计实例。
- 在测试模式下batch norm中使用的指数移动平均数只是训练统计量的线性组合，所以这些EMA只需要在检查点中保存之前进行同步。然而，batch norm的一些常见实现并不同步这些EMA，只保存第一个设备的EMA。

多主机管道的注意事项

概述： *对于记录、评估、RNG、检查点和数据分片来说，多主机训练会使其非常容易引入bug!*

- 确保管道只在一台主机上进行记录和检查点。
- 确保在评估或检查点运行之前，批量规范的统计数据在各主机之间是同步的。
- 不同主机的RNG种子是相同的（用于模型初始化），不同主机的种子是不同的（用于数据洗牌/预处理），这一点很关键，所以一定要适当地标记它们。
- 为了提高性能，通常建议将数据文件分散到各个主机上。

常见问题

最佳的学习率衰减策略是什么?

▶ *[Click to expand]*

- 这是一个开放的问题。目前还不清楚如何构建一套严格的实验来自信地回答什么是 "最佳"LR衰减策略。

- 虽然我们不知道最好策略，但我们确信，有一些（非恒定的）策略是很重要的，而且调整它也很重要。
- 在优化过程中，不同的学习率在不同的时间段效果最好。有了某种策略，模型更有可能达到一个好的学习率。

我应该使用哪种学习率衰减策略作为默认值?

▶ *[Click to expand]*

- 我们的偏好是线性衰减或余弦衰减，其他衰减系列可能也不错。

为什么有些论文采用复杂的学习率衰减策略?

▶ *[Click to expand]*

- 采用复杂学习率（LR）衰减策略的论文并不少见。
- 读者经常想知道作者是如何得出如此复杂的研究结果的。
- 许多复杂的LR衰减策略是将策略作为验证集性能的函数，以特别的方式进行调整。
    1. 用一些简单的LR衰减（或恒定的学习率）开始一个单一的训练运行。
    2. 继续进行训练，直到表现似乎停滞不前。如果发生这种情况，暂停训练。从这一点出发，用一个可能更陡峭的LR衰减策略（或更小的恒定学习率）恢复训练。重复这个过程，直到启动的最后期限。
- 轻率地复制所产生的策略通常不是一个好主意，因为最好的特定策略将对一系列其他的超参数敏感。
    - 最好是复制产生策略的算法。
- 这种对验证错误敏感的策略，如果能够完全自动化的话，使用起来是没有问题的，但是作为验证错误的函数策略是很脆弱的，不容易重现，所以我们建议避免使用。
    - 在发表使用这种策略的结果之前，请尽量使其完全可重现。

应该如何调整Adam的超参数?

▶ *[Click to expand]*

- 如上所述，对搜索空间和应该从搜索空间采样多少个点做出一般性的陈述是非常困难的。请注意，并非Adam中的所有超参数都同样重要。以下经验法则对应于研究中不同的试验数量的 "预算"。
    - 如果在一项研究中<10次试验，只调整（基础）学习率。
    - 如果实验次数在10-25，调整learning rate和β_1。
    - 如果实验次数25+，调整learning rate、β_1 和 ϵ.
    - 如果可以运行大大超过25个实验，还可以调整β_2.

为什么在调整的探索阶段使用准随机搜索而不是更复杂的黑箱优化算法?

▶ *[Click to expand]*

- 准随机搜索（基于低差异序列）作为迭代调参过程的一部分，旨在最大限度地了解调参问题（我们称之为 "探索阶段"），是我们的首选，而不是更高级的黑盒子优化工具。贝叶斯优化和类似工具更适合于开发阶段。
- 基于随机移位的低差异序列准随机搜索可以被认为是 "抖动的、洗牌的网格搜索"，因为它均匀地、但随机地探索一个给定的搜索空间，并且比随机搜索更分散搜索点。
- 与更复杂的黑箱优化工具（如贝叶斯优化、进化算法）相比，准随机搜索的优势包括：

1. 对搜索空间的非适应性采样使得在事后分析中改变调整目标成为可能，而无需重新进行实验。
    - 例如，我们通常希望找到在训练的任何时候达到最佳验证损失的试验。但是，准随机搜索的非适应性使得我们有可能根据最终的验证损失、训练损失或一些替代的评价指标来找到最佳试验，而不需要重新进行任何实验。
2. 准随机搜索的行为是一致的，在统计学上是可重复的。
    - 即使搜索算法的实施发生了变化，只要保持相同的均匀性属性，就应该可以重现六个月前的研究。如果使用复杂的贝叶斯优化软件，不同版本之间的实现方式可能会发生重要变化，从而使重现旧的搜索变得更加困难。并不总是能够回滚到一个旧的实现（例如，如果优化工具是作为服务运行的）。
3. 它对搜索空间的统一探索使其更容易推理出结果以及对搜索空间的建议。
    - 例如，如果准随机搜索中的最佳点是在搜索空间的边界上，这是一个很好的（但不是万无一失的）信号，说明搜索空间的边界应该被改变。本节将进行更深入的探讨。然而，一个自适应的黑盒优化算法可能因为一些不幸运的早期试验而忽略了搜索空间的中间部分，即使它恰好包含同样好的点，因为一个好的优化算法需要采用的正是这种非均匀性来加快搜索速度。
4. 在使用准随机搜索（或其他非适应性搜索算法）时，平行运行与顺序运行不同数量的试验不会产生统计学上的不同结果，这与适应性算法不同。
5. 更复杂的搜索算法不一定能正确处理不可行的点，特别是如果它们在设计时没有考虑到神经网络超参数的调整。
6. 准随机搜索很简单，当许多调参试验平行运行时，效果特别好。
    - 从一些轶事来看，自适应算法很难打败拥有2倍预算的准随机搜索，特别是当许多试验需要平行运行时（因此在启动新试验时很少有机会利用以前的试验结果）。
    - 如果没有贝叶斯优化和其他高级黑盒优化方法的专业知识，我们可能无法实现它们在原则上能够提供的好处。在现实的深度学习调整条件下，很难对高级黑盒优化算法进行基准测试。它们是当前非常活跃的研究领域，而且对于没有经验的用户来说，更复杂的算法也有自己的陷阱。这些方法的专家能够获得良好的结果，但在高平行度条件下，搜索空间和预算往往更重要。

- 也就是说，如果我们的计算资源只允许少量的试验并行运行，而我们可以负担得起许多试验的顺序运行，那么贝叶斯优化就会变得更有吸引力，尽管会使我们的调参结果更难解释。

我在哪里可以找到准随机搜索的实现？

▶ *[Click to expand]*

- 我们使用这种实现方式，为给定的搜索空间生成一个Halton序列（旨在实现https://arxiv.org/abs/1706.03200 中推荐的移位、加扰的Halton序列）。
- 如果没有基于低差异序列的准随机搜索算法，可以用伪随机统一搜索代替，尽管这样做的效率可能会略低。
    - 在1-2个维度中，网格搜索也是可以接受的，尽管在更高的维度中不能接受 (参考Bergstra & Bengio, 2012).

使用准随机搜索需要多少次试验才能获得好的结果？

▶ *[Click to expand]*

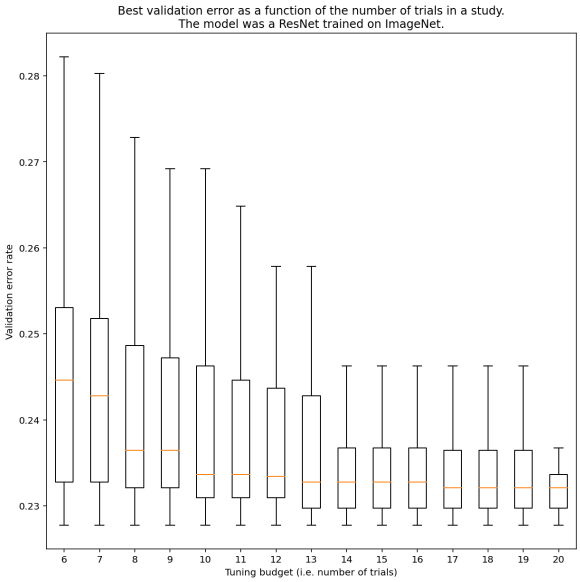

图 3: 在ImageNet上用100次试验对ResNet-50进行了调参。通过bootstrapping，模拟了不同数量的调参预算。上面绘制了每个试验预算的最佳性能箱形图。

- 没有办法笼统地回答这个问题，但我们可以看一下具体的例子。
- 如图3所示，一项研究中的试验数量对结果有很大影响。
  - 请注意，当6个试验被抽样时，与20个试验被抽样时相比，四分位数范围有多大。
  - 即使有20次试验，特别幸运和不幸运的研究之间差异也可能大于该模型在不同的随机种子上重新训练的典型变化，固定的超参数，对于这个工作量来说，可能是+/-0.1%左右，验证错误率为~23%。

如何对优化失败进行调试和缓解?

▶ *[Click to expand]*

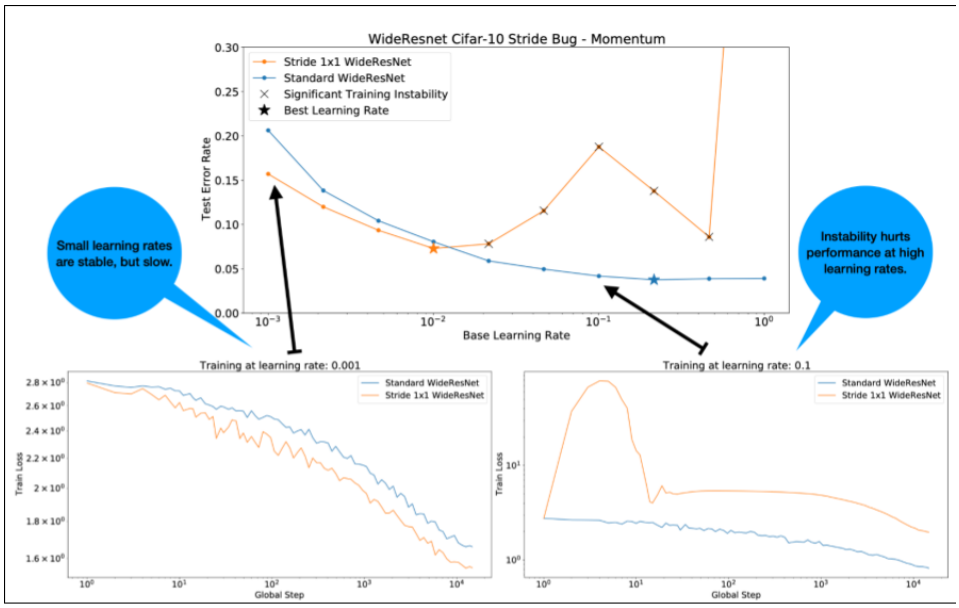

图 4: 在WideResnet中改变单个剩余块（2x2->1x1）的步长会导致训练不稳定。这不会降低低学习率下的性能，但高学习率下由于不稳定而不再训练良好。应用1000步的学习率预热解决了这一特殊的不稳定情况，允许在最大学习率为0.1时进行稳定训练。

识别不稳定的工作负载

- 如果学习率太大，任何工作负载都会变得不稳定。只有在迫使我们使用小的学习率时，不稳定才是一个问题。
- 至少有两种类型的训练不稳定性值得区分。
    1. 初始化/训练初期的不稳定性。
    2. 在训练中突然出现不稳定。
- 我们可以采取系统的方法来确定我们工作中的稳定性问题
    1. 做一次学习率扫描，找到最佳学习率lr*。
    2. 绘制学习率刚刚超过lr*的训练损失曲线。
    3. 如果学习率>lr*显示出损失的不稳定性（在训练期间损失上升而不是下降），那么修复不稳定性很可能会带来更好的训练效果。
- 在训练过程中记录全损梯度的L2准则，离群值会导致训练中间出现虚假的不稳定。这可以为如何挑选梯度/更新剪切提供参考。

注意：有些模型显示出非常早期的不稳定性，然后是恢复，趋于缓慢但稳定的训练。**普通的评估策略可能会因为评估不够频繁而错过这些问题！**

为了检查这一点，我们可以用`lr = 2 * current best`来训练一个大约500步的简略运行，但要评估每一步。

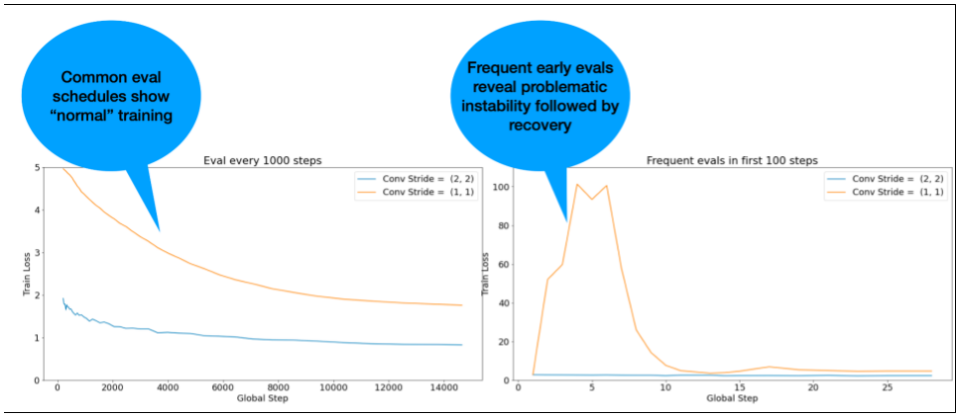

图 5: 如果怀疑模型受到早期训练不稳定的影响,在训练开始时进行频繁的评估，是很有用的。

常见不稳定模式的潜在修复方法

- 采用learning rate warmup
    - 最适合早期训练的不稳定性。
- 应用梯度剪裁
    - 对早期和中期训练的不稳定性都有好处，可能会修复一些热身运动无法修复的不良状态。
- 尝试新的优化器
    - 有时Adam可以处理Momentum不能处理的不稳定性。这是一个活跃的研究领域。
- 确保我们的模型架构使用最佳实践/初始化（下面的例子）
    - 如果模型中不包含残差连接和归一化，则添加残差连接和归一化。
- 归一化应该是残差前的最后一个操作。例如x + Norm(f(x))
- Norm(x + f(x))会导致问题。
- 尝试将剩余分支初始化为0。 (例如ReZero init).
- 降低学习率

- 这是最后的手段。

Learning rate warmup

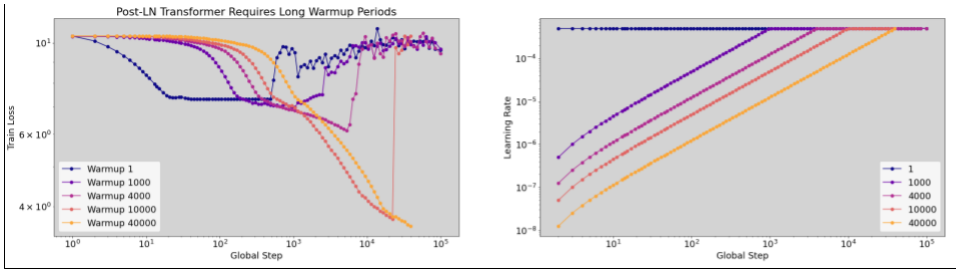

图 6: warmup期间不稳定的一个例子（注意横轴的对数比例）。在这种情况下，需要40000steps热身才能成功训练。

什么时候采用learning rate warmup

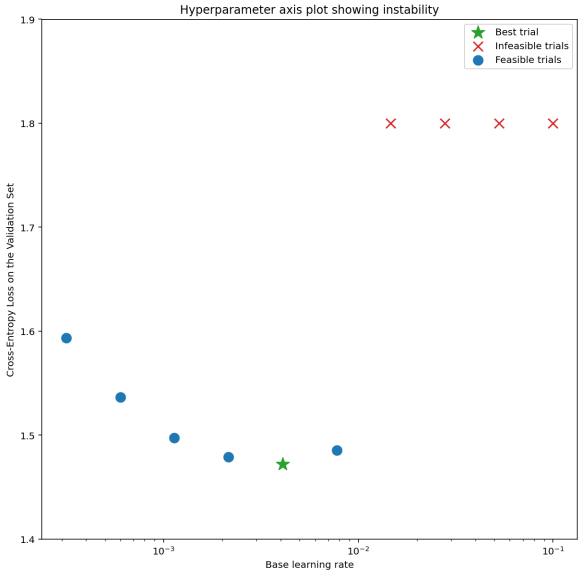

图 7a: 一个表现出训练不稳定性的模型超参数轴图的例子。最佳的学习率是在可行的边缘。一个 "不可行 "的试验被定义为产生NaN或高损失值

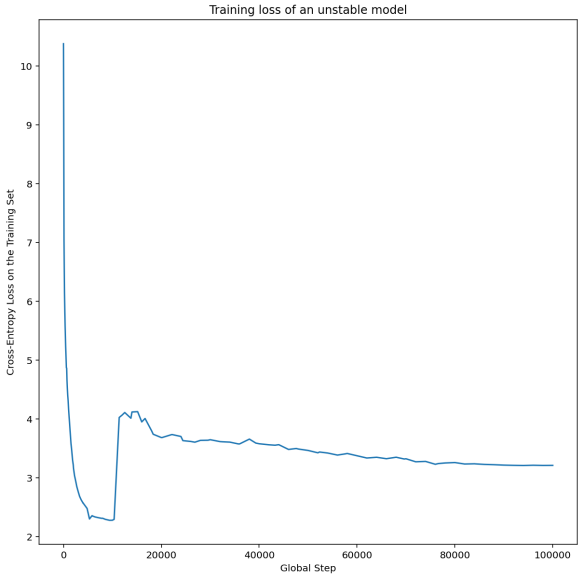

图 7b: 使用不稳定学习率模型的训练损失

- 图7a显示了一个超参数轴图，表明一个模型正在经历优化不稳定，因为最佳学习率正好处于不稳定的边缘。
- 图7b显示了如何通过检查（比峰值大5倍或10倍的学习率训练模型的训练损失）来进行双重检查。如果该图显示损失在稳步下降后突然上升（例如，在上图中的步骤~10k），那么该模型很可能患有优化不稳定。

如何应用learning rate warmup

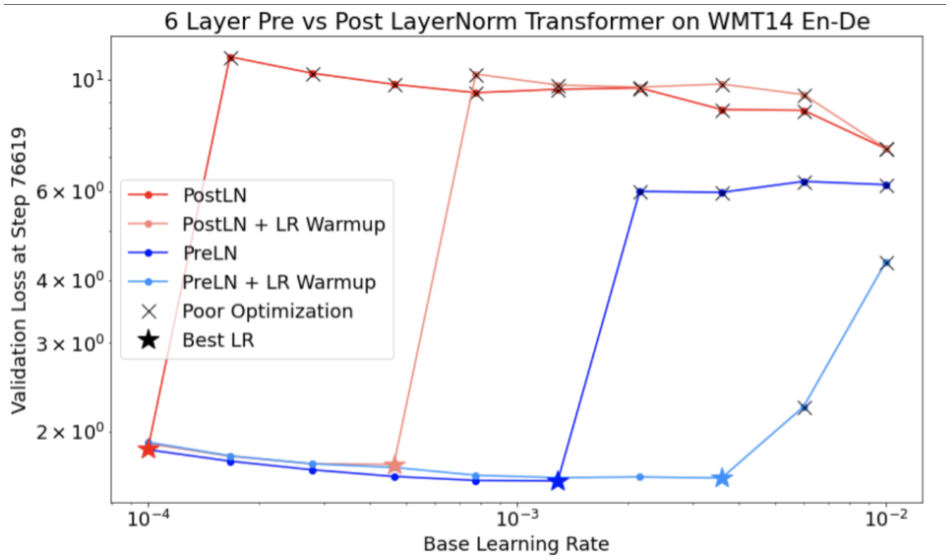

图 8: warmup对解决训练不稳定现象有很好的效果

- 利用上面的章节，我们假设实践者已经确定了模型变得不稳定的学习率。其值为`unstable_base_learning_rate`。
- Warmup包括预设一个学习率策略，将学习率从0提升到某个稳定的`base_learning_rate`，至少比`unstable_base_learning_rate`大一个数量级。默认情况下，将尝试一个10倍于`unstable_base_learning_rate`的`base_learning_rate`。尽管注意到有可能再次运行这整个过程，以获得类似100倍的效果`unstable_base_learning_rate`。具体的策略是：
  - Ramp up 从0到`base_learning_rate`经过`warmup_steps`.
  - 以恒定的速度训练`post_warmup_steps`。
- 我们的目标是找到最短的`warmup_steps`使我们能够获得远高于`unstable_base_learning_rate`的峰值学习率。
- 因此对于每个`base_learning_rate`，我们需要调整`warmup_steps`和`post_warmup_steps`。通常将`post_warmup_steps`设置为`2*warmup_steps`。
- Warmup可以独立于现有的衰变策略进行调整。`warmup_steps`应该在几个不同的数量级上进行扫描。例如，可以尝试[10, 10^3, 10^4,10^5]。最大的可行点应该不超过`max_train_steps`的10%。
- 一旦建立了一个稳健的`base_learning_rate`训练的`warmup_steps`，就应该将其应用于基线模型。从本质上讲，我们把这个策略预置到现有的策略上，并使用上面讨论的最佳检查点来比较这个试验和基线。例如，如果我们原来有10,000个`max_train_steps`并做了1000步的`warmup_steps`新的训练程序应该总共运行11000步。

- 如果稳定训练需要较长的 `warmup_steps`（> `max_train_steps` 的5%），可能需要增加 `max_train_steps` 以考虑到这一点。
- 在所有的模型中，其实并没有一个 "典型 "的数值。有些模型只需要100步，而其他模型（特别是transformer）可能需要4w步以上。

梯度裁剪

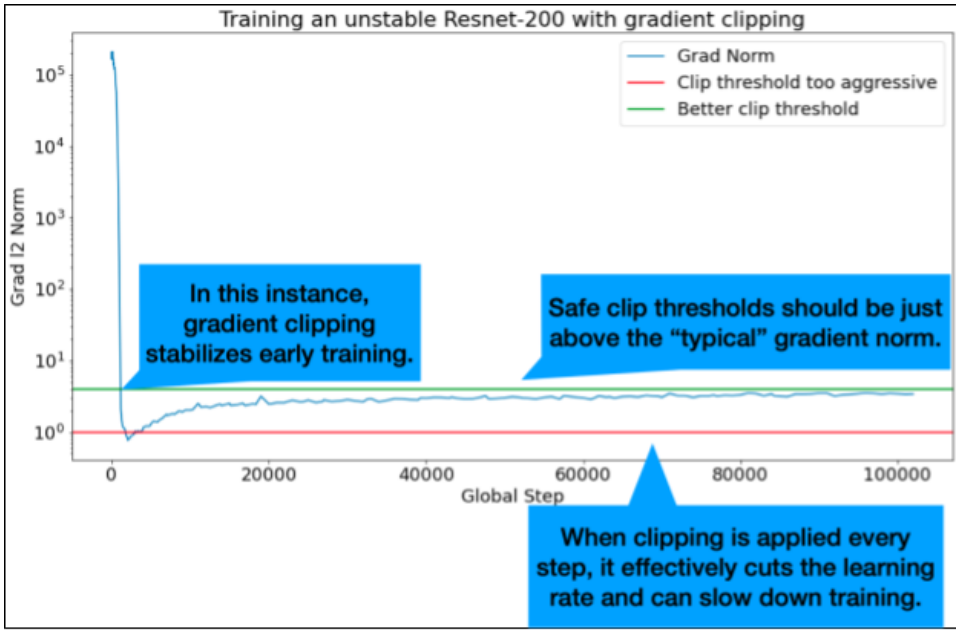

图 9: 梯度剪裁纠正早期训练不稳定性的说明。

- 当发生大的或离群的梯度问题时，梯度剪裁是最有用的。
- 剪切可以修复早期训练的不稳定性（早期的大梯度规范），或训练中期的不稳定性（训练中期的突然梯度尖峰）。
- 有时，较长的warmup可以纠正不稳定的情况，而裁剪则不能，参考本节
  - 😈 warmup时采用裁剪会怎么样?
- 理想的裁剪阈值刚刚超过 "典型 "的梯度规范。
- 下面是一个关于如何进行梯度剪裁的例子。
  - 如果梯度的规范 $|g|$ 大于梯度裁剪阈值λ， $g' = \lambda \times \frac{g}{|g|}$ 其中 g' 是新的梯度。
- 在训练过程中记录未裁剪的梯度。默认情况下生成：
  - 梯度规范与step的关系图
  - 所有steps中梯度规范的直方图
- 根据梯度规范的第90个百分点选择梯度剪裁阈值。
  - 阈值将取决于工作量，但90%是一个好的起点。如果不成功，这个阈值可以调整一下。
  - 😈 某种适应性策略呢?
- 如果我们尝试梯度剪裁，不稳定的问题仍然存在，我们可以更努力地尝试（即让阈值更小）。
- 极为激进的梯度剪裁在本质上是一种降低学习率的奇怪方式。如果我们发现自己在使用极为激进的剪裁，我们也许应该直接削减学习率。
- 我们通常会认为有>50%的更新以某种方式被剪掉是 "极其积极的"。
- 如果我们需要做极为激进的梯度剪裁来处理不稳定问题，那还不如降低学习率。

为什么你把学习率和其他优化参数称为超参数？它们不是任何先验分布的参数。

▶ *[Click to expand]*

- 诚然，"超参数 "一词在贝叶斯机器学习中有着确切的含义，将学习率和我们在深度学习中调整的大多数其他参数称为 "超参数 "是一种术语的滥用。
- 对于学习率、架构参数以及我们在深度学习中调整的所有其他东西，我们更愿意使用 "元参数 "这一术语，因为它避免了因误用 "超参数 "一词而产生的潜在混淆（在讨论贝叶斯优化时，概率响应面模型有其真正的超参数，这种混淆特别容易发生）。
- 不幸的是，尽管有可能造成混淆，但超参数这个词在深度学习界已经变得极为普遍。
- 因此，对于像本文这样一份面向广大读者的文件，其中包括许多不太可能知道这一技术问题的人，我们选择对该领域的一个混乱来源做出贡献，希望能避免另一个混乱。
- 也就是说，在发表研究论文时，我们可能会做出不同的选择，而且我们会鼓励其他人在大多数情况下使用 "元参数 "来代替。

为什么不应该调整batch size来直接提高验证集的性能?

▶ *[Click to expand]*

- 在不改变训练管道任何其他细节的情况下，改变batch size往往会影响验证集的性能。
- 然而，如果为每个batch size独立优化训练管道，两个批次规模之间验证集的性能差异通常会消失。
- 与batch size相互作用最强的超参数是优化器超参数（如学习率、动量）和正则化超参数，因此对每个batch size进行单独调整最为重要。
    - 由于样本差异，较小的batch size在训练算法中引入了更多的噪声，而这种噪声会产生正则化效应。因此，较大的批次规模可能更容易出现过拟合，可能需要更强的正则化或额外的正则化技术。
- 此外，在改变batch size时，可能需要调整训练步骤的数量。
- 一旦考虑到所有这些影响，目前没有令人信服的证据表明批次大小会影响可实现的最大验证性能(参考Shallue et al. 2018).

所有流行的优化算法更新规则是什么?

▶ *[Click to expand]*

Stochastic gradient descent (SGD)

$$\theta_{t+1} = \theta_t - \eta_t \nabla l(\theta_t)$$

Momentum

$$v_0 = 0$$

$$v_{t+1} = \gamma v_t + \nabla l(\theta_t)$$

$$\theta_{t+1} = \theta_t - \eta_t v_{t+1}$$

Nesterov

$v_0 = 0$

$v_{t+1} = \gamma v_t + \nabla l(\theta_t)$

$\theta_{t+1} = \theta_t - \eta_t(\gamma v_{t+1} + \nabla l(\theta_t)$

RMSProp

$v_0 = 1, m_0 = 0$

$v_{t+1} = \rho v_t + (1-\rho)\nabla l(\theta_t)^2$

$m_{t+1} = \gamma m_t + \frac{\eta_t}{\sqrt{v_{t+1}+\epsilon}}\nabla l(\theta_t)$

$\theta_{t+1} = \theta_t - m_{t+1}$

ADAM

$m_0 = 0, v_0 = 0$

$m_{t+1} = \beta_1 m_t + (1-\beta_1)\nabla l(\theta_t)$

$v_{t+1} = \beta_2 v_t + (1-\beta_2)\nabla l(\theta_t)^2$

$b_{t+1} = \frac{\sqrt{1-\beta_2^{t+1}}}{1-\beta_1^{t+1}}$

$\theta_{t+1} = \theta_t - \alpha_t \frac{m_{t+1}}{\sqrt{v_{t+1}}+\epsilon} b_{t+1}$

NADAM

$m_0 = 0, v_0 = 0$

$m_{t+1} = \beta_1 m_t + (1-\beta_1)\nabla l(\theta_t)$

$v_{t+1} = \beta_2 v_t + (1-\beta_2)\nabla l(\theta_t)^2$

$b_{t+1} = \frac{\sqrt{1-\beta_2^{t+1}}}{1-\beta_1^{t+1}}$

$\theta_{t+1} = \theta_t - \alpha_t \frac{\beta_1 m_{t+1} + (1-\beta_1)\nabla l(\theta_t)}{\sqrt{v_{t+1}}+\epsilon} b_{t+1}$