通过梯度下降学习梯度下降

Marcin Andrychowicz[1], Misha Denil[1], Sergio Gómez Colmenarejo[1], Matthew W. Hoffman[1],
David Pfau[1], Tom Schaul[1], Brendan Shillingford[1,2], Nando de Freitas[1,2,3]
[1]Google DeepMind [2]University of Oxford [3]Canadian Institute for Advanced Research
marcin.andrychowicz@gmail.com
{mdenil,sergomez,mwhoffman,pfau,schaul}@google.com
brendan.shillingford@cs.ox.ac.uk, nandodefreitas@google.com

摘要

在机器学习中，从人工设计的特征到通过机器学习学习特征的转变非常成功。尽管如此，优化算法仍然是手工设计的。在本文中，我们展示了如何将优化算法的设计转化为一个学习问题，允许算法学习以自动的方式探索的问题的结构。我们学习的算法由 LSTMs 实现，在训练的任务上优于一般的、手工设计的竞争对手，并且也很好地推广到具有相似结构的新任务。我们在许多任务中演示了这一点，包括简单的凸问题、训练神经网络和用神经艺术设计图像。

1 前言

通常，机器学习中的任务可以表示为优化在某个域 $\theta \in \Theta$ 上定义的目标函数的问题 $f(\theta)$。这种情况下的目标是找到最小值 $\theta^* = \arg\min_{\theta\in\Theta} f(\theta)$。尽管可以应用任何能够最小化该目标函数的方法，但是可微函数的标准方法是某种形式的梯度下降，从而导致一系列更新

$$\theta_{t+1} = \theta_t - \alpha_t \bigtriangledown f(\theta_t).$$

然而，原始梯度下降的性能受到以下事实的阻碍，即它仅利用梯度而忽略二阶信息。经典优化技术通过使用曲率信息重新调整梯度步长来纠正这种行为，通常是通过二阶偏导数的 Hessian 矩阵，尽管其他选择如广义高斯-牛顿矩阵或 Fisher 信息矩阵也是可能的。

现代优化工作的大部分都是基于设计针对特定问题类别的更新规则，不同的研究团体对不同类型的问题感兴趣。例如，在深度学习社区中，我们看到了大量专门针对高维非凸优化问题的优化方法。其中包括 momentum[Nesterov, 1983, Tseng, 1998]，Rprop [Riedmiller and Braun, 1993]，Adagrad [Duchiet al., 2011]，RMSprop [Tieleman and Hinton, 2012]，和 ADAM [Kingma and Ba, 2015]。当知道优化问题的更多结构时，也可以应用更有针对性的方法 [Martens and Grosse, 2015]。相比之下，关注稀疏性的社区倾向于采用截然不同的方法 [Donoho, 2006, Bach et al., 2012]。对于松弛经常是常态的组合优化来说，情况更是如此 [Nemhauser and Wolsey, 1988]。

这个优化器设计的行业允许不同的社区创建优化方法，这些方法利用他们感兴趣的问题中的结构，代价是在该范围之外的问题上可能表现不佳。此外，最优化的 “没有免费的午餐定理” [Wolpert 和 Macready，1997] 表明，在组合优化的环境中，没有任何算法能够比期望中的随机策略做得更好。这表明问题子类的专门化实际上是提高性能的唯一途径。

在这项工作中，我们采取了一种不同的策略，建议用一个学习过的更新规则代替手工设计的更新规则，我们称之为优化器 g，由它自己的参数集 Φ 指定。这将导致以下方式更新优化 f

$$\theta_{t+1} = \theta_t - g_t(\bigtriangledown f(\theta_t), \Phi). \tag{1}$$

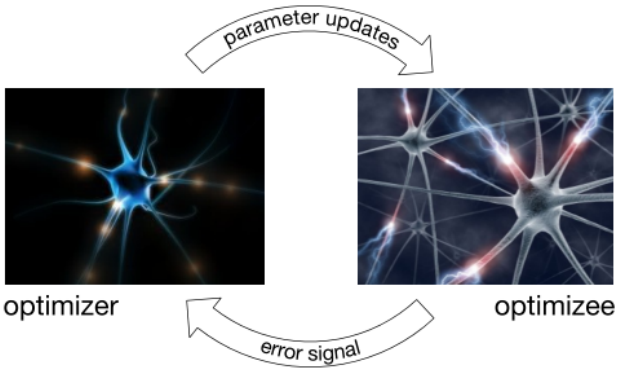

图 1: 优化器 (左) 具有优化器 (右) 的性能，并建议更新以提高优化器的性能。[照片:Bobolas, 2009, Maley, 2011]

这个过程的高层视图如图 1 所示。在接下来的内容中，我们将使用递归神经网络 (RNN) 对更新规则 g 进行显式建模，该神经网络保持其自身的状态，因此根据其迭代进行动态更新。

1.1 迁移学习和泛化

这项工作的目标是开发一个程序来构建一个学习算法，在一类特定的优化问题上表现良好。将算法设计作为一个学习问题，允许我们通过示例问题实例来指定我们感兴趣的问题类别。这与通过分析来描绘感兴趣问题的特性并使用这些分析见解来手工设计学习算法的普通方法形成对比。

在这个框架中考虑概括的意义是有益的。在普通的统计学习中，我们有一个特别感兴趣的函数，它的行为通过一组示例函数评估来约束。在选择模型时，我们指定了一组归纳偏差，这些偏差是关于我们认为感兴趣的函数在我们没有观察到的点上应该如何表现，而泛化对应于在新的点上对目标函数的行为做出预测的能力。在我们的设置中，例子本身就是问题实例，这意味着泛化对应于在不同问题之间传递知识的能力。这种问题结构的重用通常被称为迁移学习，并且通常被视为一门独立的学科。然而，从元学习的角度来看，我们可以把迁移学习的问题归结为一个泛化问题，这个问题在机器学习领域得到更深的研究。

1.2 简史及相关工作

使用学习来学习或元学习来获取知识或归纳偏见的想法由来已久 [Thrun and Pratt, 1998]。最近，Lake et al.[2016] 有力地论证了它作为人工智能构件的重要性。类似地，Santoro et al. [2016 将多任务学习框架视为泛化，然而与我们的方法不同，它们直接训练基础学习者，而不是训练算法。一般而言之，这些想法涉及在两个不同的时间尺度发生的学习：在许多不同任务中的快速学习和更加渐进的元学习。

也许最普遍的元学习方法是 Schmidhuber[1992, 1993] 的方法——[Schmidhuber, 1987] 的基础上——它认为网络能够修改自己的权重。这样的系统是端到端可微的，允许网络和学习算法在很少的限制下通过梯度下降联合训练。然而，这种通用性的代价是使学习规则变得非常难以训练。另一方面，Schmidhuber 等人 [1997] 使用 Success Story 算法来修改其搜索策略，而不是梯度下降;Daniel 等人最近也采用了类似的方法 [2016]，该方法使用强化学习来训练控制器选择步长。Bengio 等人 [1990,1995] 提出使用简单的参数规则来学习避免反向传播的更新。就本文的重点而言，Bengio 等人的工作可以被描述为通过梯度下降来学习没有梯度下降的学习。Runarsson 和 Jonsson[2000] 的工作建立在这一工作的基础上，用神经网络代替了简单的规则。

Cotter 和 Conwell[1990] 以及后来的 Younger 等人 [1999] 也证明了固定权值的循环神经网络可以在不需要修改网络权值的情况下表现出动态行为。类似地，在过滤上下文中也显示了这一点 (如。Feldkamp 和 Puskorius, 1998]，这与简单的多时间尺度优化器直接相关 [Sutton, 1992, Schraudolph, 1999]。

最后，Younger 等人 [2001] 和 Hochreiter 等人 [2001] 的工作将这些不同的研究线索连接起来，允许从一个网络的反向传播输出馈入另一个学习网络，这两个网络共同训练。我们的元学习方法建立在这项工作的基础上，通过修改优化器的网络结构，以便将这种方法扩展到更大的神经网络优化问题。

2 学习用递归神经网络学习

在这项工作中，我们考虑直接参数化优化器。因此，在稍微滥用符号的情况下，我们将根据优化器参数 $\theta^*(f, \Phi)$ 和所讨论的函数来编写最终的优化器参数 Φ。然后我们可以问这样一个问题: 优化器好意味着什么？给定函数 f 的分布，我们将把预期损失写成

$$\mathcal{L}(\Phi) = \mathbb{E}_f[f(\theta^*(f, \Phi))] \tag{2}$$

如前所述，我们将把更新步骤 g_t 作为递归神经网络 m 的输出，由 Φ 参数化，其状态我们将用 h_t 显式表示。接下来，虽然 (2) 中的目标函数只取决于最终的参数值，但为了训练优化器，对于某个水平 T，可以方便地有一个依赖于优化的整个轨迹的目标。

$$\mathcal{L}(\Phi) = \mathbb{E}_f\left[\sum_{t=1}^{T} w_t f(\theta_t)\right] \qquad where \qquad \begin{aligned} \theta_t + 1 &= \theta_t + g_t, \\ \begin{bmatrix} g_t \\ h_t + 1 \end{bmatrix} &= m(\bigtriangledown_t, h_t, \Phi). \end{aligned} \tag{3}$$

这里 $w_t \in \mathbb{R}_{\geq 0}$ 是与每个时间步相关联的任意权重，我们也将使用符号 $\bigtriangledown_t = \bigtriangledown_\theta f(\theta_t)$。当 $w_t = 1[t = T]$ 时，这个公式等价于 (2)，但稍后我们将描述为什么使用不同的权值可以证明是有用的。

我们可以通过对 Φ 进行梯度下降来最小化 $\mathcal{L}(\Phi)$ 的值。梯度估计 $\frac{\partial \mathcal{L}(\Phi)}{\partial \Phi}$ 可以通过对随机函数 f 采样并对图 2 中的计算图形应用反向传播来计算。我们允许梯度沿图形中的实心边缘流动，但沿虚线边缘的梯度被丢弃。忽略沿虚线边缘的梯度相当于假设优化器的梯度不依赖于优化器参数，即 $\partial \bigtriangledown_t / \partial \Phi = 0$。这一假设使我们可以避免计算 f 的二阶导数。

检查 (3) 中的目标，我们看到梯度只有在 $w_t \neq 0$ 项时才非零。如果我们使用 $w_t = 1[t = T]$ 来匹配原问题，那么轨迹前缀的梯度为零，只有最后的优化步骤提供了训练优化器的信息。这使得时间反向传播 (Backpropagation Through Time, BPTT) 效率低下。我们通过放松目标来解决这个问题，使 $w_t > 0$ 在沿轨迹的中间点处。这改变了目标函数，但允许我们在部分轨迹上训练优化器。为了简单起见，在所有的实验中，我们对每个 t 都使用 $w_t = 1$。

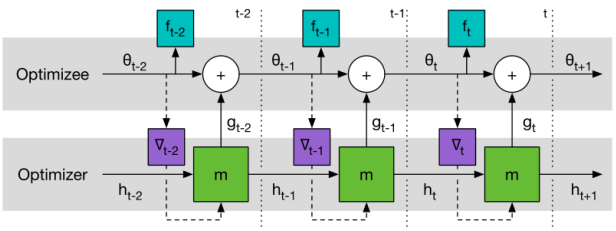

图 2: 用于计算优化器梯度的计算图

2.1 Coordinatewise LSTM 优化器

在我们的设置中应用 RNNs 的一个挑战是，我们希望能够优化至少数万个参数。用一个完全连接的 RNN 在这种规模下进行优化是不可行的，因为它需要一个巨大的隐藏状态和大量的参数。为了避免这个困难，我们将使用一个优化器 m，它协调地操作目标函数的参数，类似于其他常见的更新规则，如 RMSprop 和 ADAM。这种协调网络架构允许我们使用一个非常小的网络来定义优化器，并在优化器的不同参数之间共享优化器参数。

通过对每个目标函数参数使用单独的激活，在每个坐标上实现不同的行为。除了允许我们为这个优化器使用一个小的网络之外，这种设置还具有使优化器对网络中的参数顺序保持不变的良好效果，因为在每个坐标上独立使用相同的更新规则。

我们使用两层 Long Short Term Memory (LSTM) 网络实现每个坐标的更新规则 [Hochreiter and Schmidhuber, 1997]，使用目前标准的遗忘门体系结构。该网络将单个坐标的最优梯度和之前的隐藏状态作为输入，输出相应的最优参数的更新。我们将把这个体系结构 (如图 3 所示) 称为 LSTM 优化器。

递归的使用允许 LSTM 学习动态更新规则，这些规则集成了来自梯度历史的信息，类似于动量。众所周知，这在凸优化中有许多理想的性质 [参见 Nesterov, 1983]，事实上，许多最近的学习程序 (如 adam) 在它们的更新中使用动量。

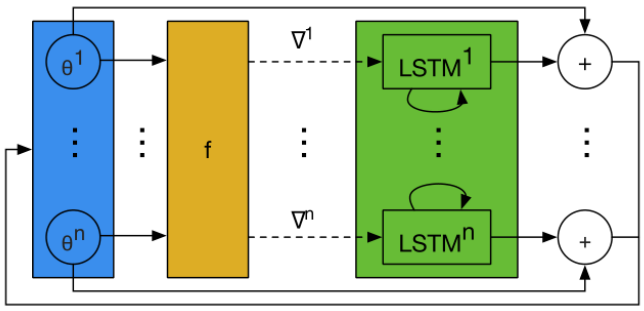

图 3: LSTM 优化器的一个步骤。所有 LSTM 都有共享的参数，但隐藏状态是独立的。

预处理和后处理 优化器的输入和输出可以有非常不同的大小取决于被优化的函数的类别，但神经网络通常只对既不是非常小也不是非常大的输入和输出稳定工作。在实践中，使用适当的常量 (在所有时间步长和函数 f 之间共享) 重新调整 LSTM 优化器的输入和输出就足以避免这个问题。在附录 A 中，我们提出了一种对优化器输入进行预处理的不同方法，这种方法更健壮，并提供了更好的性能.

3 实验

在所有实验中，训练过的优化器使用两层 LSTM，每层有 20 个隐藏单元。每个优化器都是通过使用第 2 节中描述的截断的 BPTT 来最小化等式 3 来训练的。该算法采用 ADAM 算法进行最小化，并通过随机搜索选择学习率。

我们在训练优化器时使用早期停止，以避免过拟合优化器。在每个阶段 (一些固定数量的学习步骤) 之后，我们冻结优化器参数并评估其性能。我们选择最好的优化器 (根据最终验证损失)，并报告它在一些新抽样测试问题上的平均性能。

我们将我们训练过的优化器与深度学习中使用的标准优化器进行了比较:SGD、RMSprop、ADAM 和 Nesterov 的加速梯度 (NAG)。对于每一个优化器和每一个问题，我们都调整了学习速率，并以给出每个问题的最佳最终错误

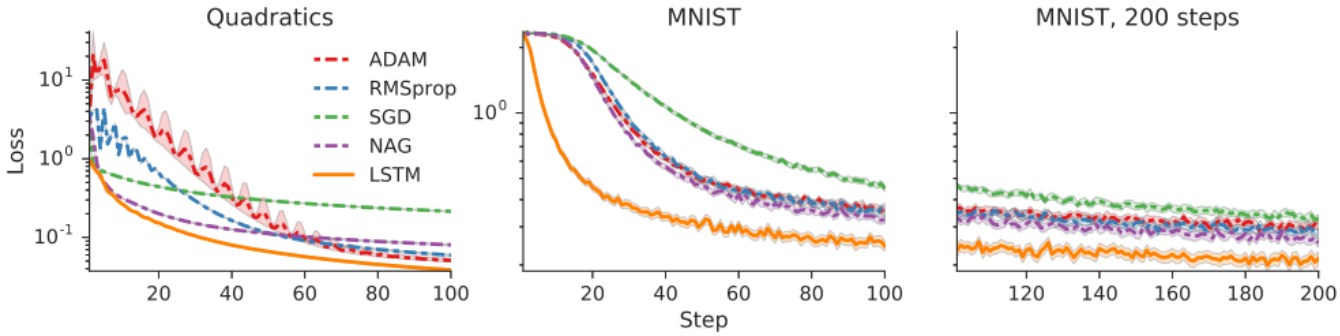

图 4: 比较学习型和手工优化器的性能。学习过的优化器用实线显示，手工制作的优化器用虚线显示。MNIST 图中 y 轴的单位为对数。左图: 不同优化器对随机抽样的 10 维二次函数的性能。中间:LSTM 优化器优于在 MNIST 上训练基网络的标准方法。右图: 通过一个经过训练的优化器学习 100-200 步的曲线 (中心图的延续)

的速率报告结果。当优化器拥有比学习速率更多的参数时 (例如 ADAM 的衰减系数)，我们将使用 Torch7 中 optim 包的默认值。所有优化参数的初始值从 IID 高斯分布采样。

3.1 二次函数（Quadratic functions）

在这个实验中，我们考虑在一类简单的合成 10 维二次函数上训练优化器。特别地，我们考虑最小化函数的形式

$$f(\theta) = \|W\theta - y\|_2^2$$

对于不同的 10x10 矩阵 W 和 10 维向量 y，它们的元素来自于 IID 高斯分布。优化人员通过优化这类随机函数进行训练，并在来自相同分布的新抽样函数上进行测试。每个功能优化 100 步，训练过的优化人员展开 20 步。我们没有使用任何预处理或后处理。

不同优化器的学习曲线，在许多函数上取平均值，如图 4 的左图所示。每条曲线对应一个优化算法在多个测试函数上的平均性能; 实曲线表示学习到的优化器性能，虚线曲线表示标准基线优化器的性能。很明显，在这个设置中，学习过的优化器的性能大大优于基线。

3.2 在 MNIST 上训练一个小型神经网络

在这个实验中，我们测试了可训练的优化器是否可以学习在 MNIST 上优化一个小型神经网络，也探索了训练过的优化器如何泛化到它们训练过的之外的功能。为此，我们训练优化器来优化基础网络，并在测试时探索对网络架构和训练程序的一系列修改。

在此设置中，目标函数 $f(\theta)$ 是带有参数 θ 的小 MLP 的交叉熵。f 的值和梯度 $\partial f(\theta)/\partial\theta$ 是使用 128 个例子的随机小批估计的。基本网络是一个 MLP，其隐含层为 20 个单元，使用 sigmoid 激活函数。不同运行之间可变性的唯一来源是初始值 θ_0 和小批量选择中的随机性。每次优化运行 100 步，训练过的优化器展开 20 步。我们采用附录 A 中描述的输入预处理，并将 LSTM 的输出按 0.1 的系数进行缩放。

图 4 的中心图显示了使用不同优化器的基本网络的学习曲线。在这个实验中，NAG、ADAM 和 RMSprop 表现出大致相同的性能，LSTM 优化器的性能明显优于它们。如果允许 LSTM 优化器运行 200 步 (尽管它已经被训练为优化 100 步)，图 4 中的正确图将比较它的性能。在这个比较中，我们重用了之前实验中的 LSTM 优化器，在这里我们看到 LSTM 优化器在这个任务上依然优于基线优化器。

对不同结构中的泛化 图 5 显示了三个应用 LSTM 优化器训练网络的例子，这些网络的结构不同于训练它的基础网络。这些修改是 (从左到右)(1)MLP 的 40 个隐藏单元而不是 20 个，(2) 网络的两个隐藏层而不是一个，

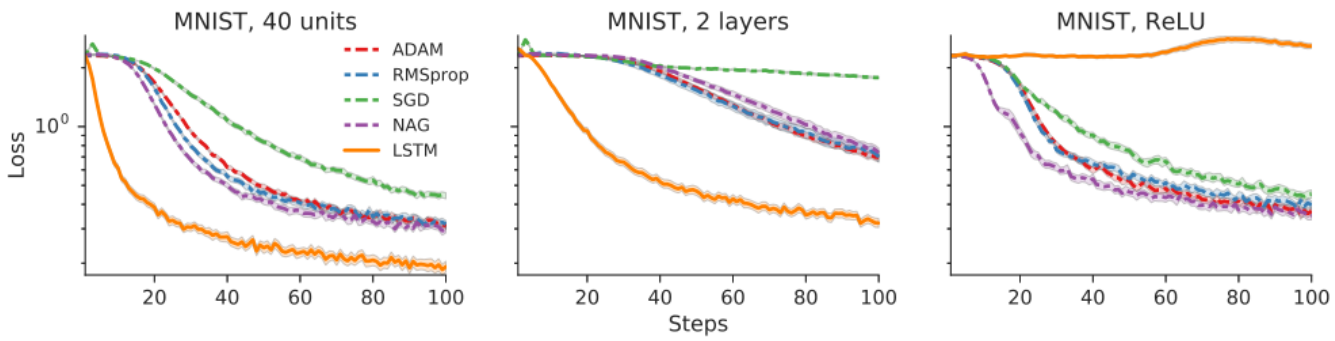

图 5: 学习型和手工优化器性能的比较。y 轴的单位是对数。左: 推广到不同数量的隐藏单位 (40 而不是 20)。中心: 推广到不同数量的隐藏层 (2 层而不是 1 层)。这个优化问题很难，因为隐藏层很窄。右图: 用 ReLU 激活一个有 20 个隐藏单位的 MLP 的训练曲线。LSTM 优化器是在带有 sigmoid 激活的 MLP 上训练的。

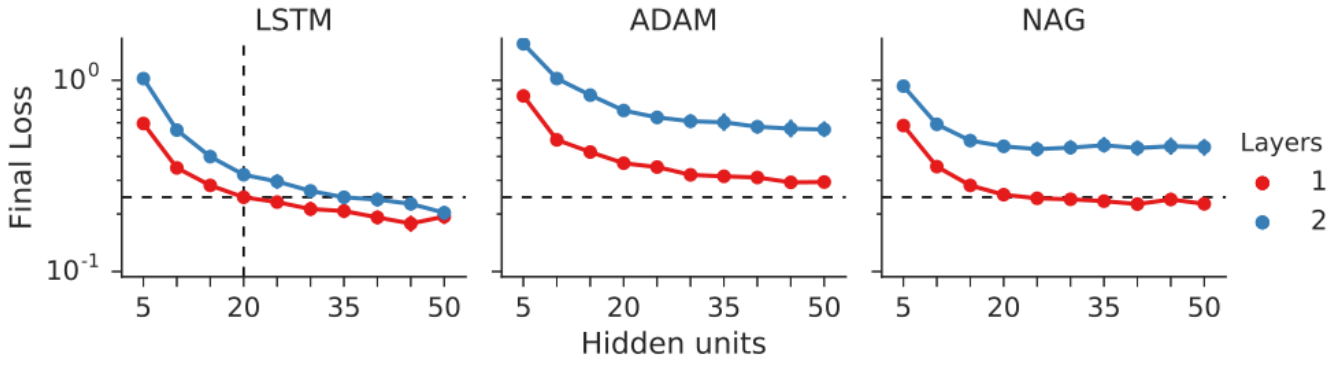

图 6: 随着优化架构的变化，使用 sigmoid 非线性对最终 MNIST 性能进行系统研究。最左侧图中的垂直虚线表示训练 LSTM 的架构，水平线表示在此设置下训练优化器的最终性能。

(3) 网络使用 ReLU 激活而不是 sigmoid。在前两种情况下，LSTM 优化器泛化良好，并且继续优于手工设计的基线，尽管在其训练机制之外运行。然而，将激活函数更改为 ReLU 会使学习过程的动态充分不同，使学习过的优化器不再能够泛化。最后，在图 6 中，我们展示了系统地改变测试体系结构的结果; 对于 LSTM 结果，我们再次使用使用 1 层 20 个单元和 sigmoid 非线性训练的优化器。请注意，在这种测试集问题与训练集中的问题非常相似的情况下，我们看到了比基线优化器更好的泛化能力。

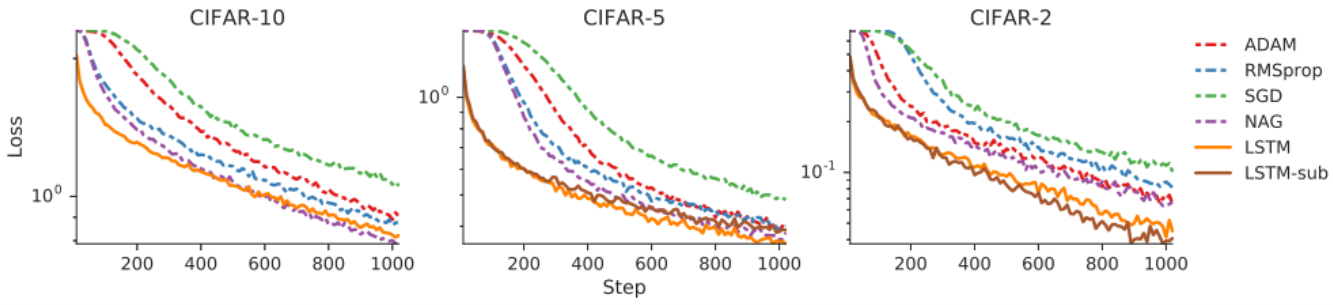

图 7: 在 CIFAR-10 数据集和子集上的优化性能。左边显示的是 LSTM 优化器与在 CIFAR-10 上训练并在测试集上测试的各种基线的对比。右边的两个图是这些优化器在 CIFAR 标签子集上的性能。额外的优化器 LSTM-sub 只在 heldout 标签上进行了训练，因此正在转移到一个全新的数据集。

3.3 在 CIFAR-10 上训练卷积网络

接下来，我们测试了经过训练的神经优化器在优化 CIFAR-10 数据集分类性能方面的性能 [Krizhevsky, 2009]。在这些实验中，我们使用了一个包含卷积层和前馈层的模型。特别是，这些实验中使用的模型包括 3 个最大池化的卷积层，然后是 32 个隐藏单元的全连接层; 所有的非线性都是 ReLU 激活与批归一化。

在 2.1 节中介绍的协调网络分解 (在前面的实验中使用) 利用了一个单独的 LSTM 体系结构，对于每个优化参数，它具有共享的权值，但隐藏状态是独立的。我们发现，由于完全连接层和卷积层之间的差异，这种分解对于本节中介绍的模型体系结构是不够的。相反，我们通过引入两个 lstm 来修改优化器: 一个为全连接层提出参数更新，另一个更新卷积层参数。与之前的 LSTM 优化器一样，我们仍然使用具有共享权值和单个隐藏状态的协调分解，但是 LSTM 权值现在只在相同类型的参数之间共享 (即完全连接 vs. 卷积)。

与基线技术相比，这个经过训练的优化器的性能如图 7 所示。最左边的图显示使用优化器将分类器适合于保留的测试集的结果。右边的另外两个图显示了经过训练的优化器对仅包含标签子集的修改数据集的性能，即 CIFAR-2 数据集只包含 10 个标签中的 2 个对应的数据。此外，我们还包括了一个优化器 LSTM-sub，它只在保留的标签上进行了训练。

在所有这些示例中，我们可以看到 LSTM 优化器比基线优化器学习速度快得多，CIFAR-5 特别是 CIFAR-2 数据集的性能得到了显著提高。我们还可以看到，只在不相交的数据子集上训练过的优化器几乎不会受到这种差异的影响，它们可以很好地传输到额外的数据集。

3.4 神经艺术 (Neural Art)

最近关于使用卷积网络或神经艺术的艺术风格转移的工作 [Gatys 等人，2015] 为我们的方法提供了一个自然的测试平台，因为每个内容和风格图像对都会产生不同的优化问题。每个神经艺术问题从一个内容图像 c 和一个风格图像 s 开始，由下式给出

$$f(\theta) = \alpha \mathcal{L}_content(c, \theta) + \beta \mathcal{L}_style(s, \theta) + \gamma \mathcal{L}_reg(\theta)$$

f 的最小值是样式化的图像 (styled image)。前两个术语试图将样式化图像的内容和样式与它们的第一个参数相匹配，第三个术语是一个正则化器，它鼓励样式化图像的平滑性。详情见 [Gatys et al.，2015]。

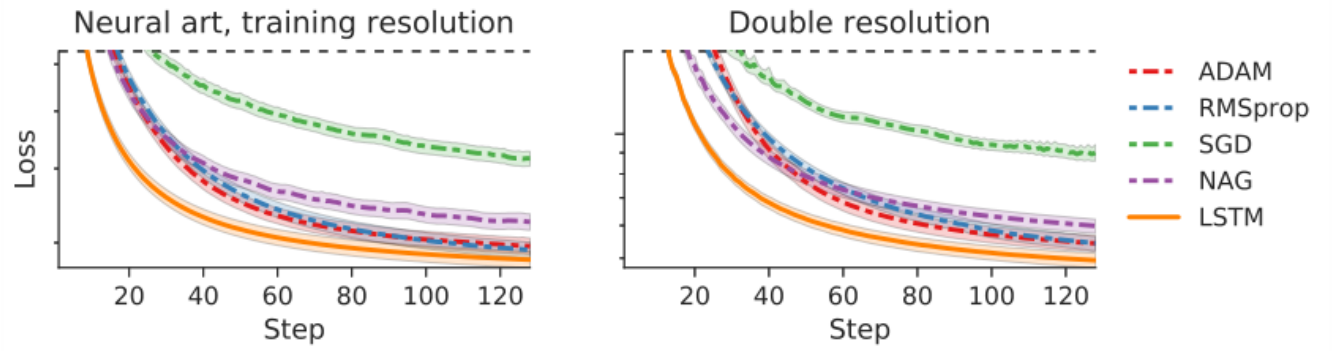

图 8: 神经艺术的优化曲线。内容图像来自测试集，在 LSTM 优化器训练中没有使用。注意:y 轴是对数尺度的，我们放大图中有趣的部分。左图: 在训练决议中应用训练风格。右图: 将测试风格应用于双倍的训练分辨率。

图 9: 使用 LSTM 优化器设计的图像示例。每个三元组由内容图像 (左)、样式 (右) 和 LSTM 优化器生成的图像 (中) 组成。左图: 以训练分辨率将训练风格应用于测试图像的结果。右图: 以两倍于优化器训练分辨率的分辨率对测试图像应用新样式的结果。

我们只使用从 ImageNet 拍摄的 1 种风格和 1800 张内容图像训练优化器 [Deng 等人，2009]。我们随机选取 100 张内容图像进行测试，20 张内容图像进行训练优化器的验证。我们训练优化器使用来自 ImageNet 的 64x64 内容图像和一个固定样式的图像。然后我们测试它如何更好地概括为不同风格和更高分辨率 (128 × 128) 的图像。每个图像优化 128 步，训练好的优化人员展开 32 步。图 9 显示了使用 LSTM 优化器样式化两个不同图像的结果。LSTM 优化器使用附录 A 中描述的输入预处理，而不使用后处理。请参阅附录 C 以获得更多图像。

图 8 比较了 LSTM 优化器和标准优化算法的性能。如果分辨率和样式图像与它所训练的图像相同，LSTM 优化器的性能优于所有标准优化器。此外，当分辨率和样式在测试时都发生改变时，它仍然可以很好地执行。

最后，在附录 B 中，我们定性地检查了由学习过的优化器生成的步骤方向的行为。

4 结论

我们已经展示了如何将优化算法的设计转换为一个学习问题，这使我们能够训练专门针对特定函数类的优化器。我们的实验已经证实，学习过的神经优化器比深度学习中使用的最先进的优化方法更好。我们见证了显著的迁移程度，例如，LSTM 优化器在 12288 个参数的神经艺术任务上训练，能够同时推广到具有 49152 个参数、不同风格和不同内容图像的任务。在 MNIST 任务中，当转移到不同的架构时，我们观察到了类似的令人印象深刻的结果。

对 CIFAR 图像标记任务的结果表明，LSTM 优化器在转换到来自相同数据分布的数据集时优于手工设计的优化器。

参考文献

F. Bach, R. Jenatton, J. Mairal, and G. Obozinski. Optimization with sparsity-inducing penalties. Foundations and Trends in Machine Learning, 4(1):1–106, 2012.