# PROXYLESSNAS:直接神经架构搜索目标任务和硬件

Han Cai, Liengeng Zhu, Song Han麻省理
工学院 {hancai, liengeng, songhan}@mit.edu

## 抽象的

神经架构搜索 (NAS) 通过自动设计有效的神经网络架构产生了巨大的影响。然而,传统 NAS 算法的高计算需求（例如 104 GPU 小时)使得很难直接在大规模任务（例如 ImageNet)上搜索架构。可区分的 NAS 可以通过网络架构的连续表示来降低 GPU 小时数的成本,但会遇到 GPU 内存消耗高的问题（根据候选集大小线性增长）。因此,他们需要利用代理任务,例如在较小的数据集上进行训练,或者只学习几个块,或者只训练几个 epoch。这些在代理任务上优化的架构并不能保证在目标任务上是最优的。在本文中,我们提出了可以直接学习大规模目标任务和目标硬件平台架构的 ProxylessNAS。我们解决了可微分 NAS 的高内存消耗问题,并将计算成本（GPU 小时数和 GPU 内存)降低到与常规训练相同的水平,同时仍然允许大的候选集。 CIFAR-10 和 ImageNet 上的实验证明了直接性和专业化的有效性。在 CIFAR-10 上,我们的模型仅用 5.7M 个参数就实现了 2.08% 的测试错误,优于之前最先进的架构 AmoebaNet-B,同时使用的参数减少了 6 倍。在 ImageNet 上,我们的模型实现了比 MobileNetV2 高 3.1% 的 top-1 精度,同时在测量的 GPU 延迟下速度提高了 1.2 倍。我们还应用 ProxylessNAS 为具有直接硬件指标（例如延迟)的硬件专门化神经架构,并为高效的 CNN 架构设计提供见解。 1

## 1简介

神经架构搜索 (NAS) 在为各种深度学习任务（例如图像识别)自动化神经网络架构设计方面取得了很大成功（Zoph 等人， 2018 年； Cai 等人， 2018a； Liu 等人， 2018a； Zhong et al., 2018)和语言建模(Zoph & Le, 2017)。尽管取得了显著成果,但传统的 NAS 算法的计算密集度过高,需要在单个实验中针对目标任务训练数千个模型。因此,直接将 NAS 应用于大规模任务（例如 ImageNet)在计算上是昂贵的或不可能的,这使得难以产生实际的行业影响。作为权衡， Zoph 等人。 (2018)建议搜索代理任务的构建块,例如训练更少的时期,从较小的数据集（例如 CIFAR-10)开始,或者用更少的块学习。然后将表现最好的块堆叠起来并转移到大规模目标任务中。该范式在后续的 NAS 算法中被广泛采用(Liu et al., 2018a;b; Real et al., 2018; Cai et al., 2018b; Liu et al., 2018c; Tan et al., 2018; Luo et al.等人， 2018 年）。

然而,这些在代理任务上优化的块并不能保证在目标任务上是最优的,尤其是在考虑延迟等硬件指标时。更重要的是,为了实现可迁移性,此类方法只需要搜索几个架构图案,然后重复堆叠相同的图案,这限制了块的多样性,从而损害了性能。

在这项工作中,我们针对上述限制提出了一种简单有效的解决方案,称为 ProxylessNAS,它直接学习目标任务和硬件上的架构,而不是通过

[1] 预训练模型和评估代码发布在https://github.com/MIT-HAN-LAB/ProxylessNAS。


图 1:ProxylessNAS 在目标任务和硬件上直接优化神经网络架构。受益于直接性和专业化,ProxylessNAS 可以取得比以前基于代理的方法更好的结果。在 ImageNet 上,仅用 200 个 GPU 小时（比 MnasNet 少 200 倍（Tan 等人， 2018 年））,我们搜索的移动 CNN 模型达到了与 MobileNetV2 1.4 相同的 top-1 精度水平,同时速度提高了 1.8 倍。

代理（图1）。我们还删除了以前 NAS 作品中重复块的限制（Zoph 等人， 2018 年； Liu 等人， 2018c） ,并允许学习和指定所有块。为实现这一目标,我们通过以下方式将架构搜索的计算成本（GPU 小时数和 GPU 内存)降低到与常规训练相同的水平。

GPU 小时数 显存

GPU hour-wise,受最近作品的启发（Liu et al., 2018c; Bender et al., 2018）,我们将 NAS 制定为路径级修剪过程。具体来说,我们直接训练一个包含所有候选路径的过参数化网络（图2）。在训练过程中,我们明确引入架构参数以了解哪些路径是冗余的,而这些冗余路径在训练结束时被修剪以获得紧凑的优化架构。这样,在架构搜索过程中,我们只需要训练一个没有任何元控制器（或超网络)的单一网络。

然而,天真地包含所有候选路径会导致 GPU 内存爆炸（Liu et al., 2018c; Bender et al., 2018）,因为内存消耗随选择数量线性增长。

普通列车 NAS 需要元控制器 飞镖和一击 没有元控制器 无代理（我们的） 没有元控制器

因此,在 GPU 内存方面,我们将架构参数（1 或 0)二值化,并强制只有一条路径在运行时处于活动状态。这将所需内存减少到与训练紧凑模型相同的水平。我们提出了一种基于梯度的方法来训练这些基于 Bina ryConnect 的二值化参数（Courbariaux 等人， 2015 年）。此外,为了处理在目标硬件上学习专用网络架构的不可微硬件目标（以延迟为例） ,我们将网络延迟建模为连续函数,并将其优化为正则化损失。此外,我们还提出了一种基于强化(Williams, 1992)的算法作为处理硬件指标的替代策略。

在我们对 CIFAR-10 和 ImageNet 的实验中,受益于直接性和专业性,我们的方法可以获得强大的实证结果。在 CIFAR-10 上,我们的模型仅用 5.7M 参数就达到了 2.08% 的测试误差。在 ImageNet 上,我们的模型达到了 75.1% 的 top-1 准确率,比 MobileNetV2 （Sandler 等人， 2018 年)高出 3.1%,同时速度提高了 1.2 倍。我们的贡献可以总结如下：

- ProxylessNAS 是第一个在没有任何代理的情况下直接在大规模数据集（例如ImageNet)上学习架构的NAS 算法,同时仍然允许大型候选集并消除重复块的限制。它有效地扩大了搜索空间并取得了更好的性能。

- 我们为 NAS 提供了一个新的路径级修剪视角,显示了 NAS 与模型压缩之间的密切联系（Han 等人， 2016 年）。通过使用路径级二值化,我们节省了一个数量级的内存消耗。 · 我们提出了一种新的基于梯度的方法（延迟正则化损失)来处理硬件目标（例如延迟）。给定不同的硬件平台:CPU/GPU/Mobile,ProxylessNAS 支持硬件感知的神经网络专业化,它针对目标硬件进行了精确优化。据我们所知,这是第一项针对不同硬件架构研究专用神经网络架构的工作。

- 大量实验显示了ProxylessNAS 的直接性和专业化特性的优势。它在不同硬件平台（GPU、CPU 和手机)的延迟限制下,在 CIFAR-10 和 ImageNet 上实现了最先进的精度性能。我们还分析了专门针对不同硬件平台的高效 CNN 模型的见解,并提高了人们对不同硬件架构需要专门的神经网络架构以进行高效推理的认识。

## 2相关工作

使用强化学习或神经进化等机器学习技术代替人类专家设计神经网络架构（通常称为神经架构搜索)引起了越来越多的兴趣（Zoph & Le, 2017; Liu et al., 2018a;b;c; Cai et al., 2018a;b; Pham et al., 2018; Brock et al., 2018; Bender et al., 2018; Elsken et al., 2017; 2018b; Ka math等人， 2018 年）。在 NAS 中,架构搜索通常被认为是一个元学习过程,并且引入了一个元控制器（例如递归神经网络（RNN）)来探索给定的架构空间,并在内部循环中训练网络以获得评价指导探索。因此,运行此类方法的计算成本很高,尤其是在大规模任务上,例如 ImageNet。

最近的一些工作（Brock 等人， 2018 年； Pham 等人， 2018 年)试图通过降低获得评估的成本来提高这种元学习过程的效率。在布罗克等人。 (2018),一个超网络被用来为每个采样网络生成权重,因此可以在不训练它的情况下评估架构。同样， Pham 等人。 (2018)建议在标准 NAS 框架下共享所有样本网络的权重(Zoph & Le, 2017)。这些方法将架构搜索速度提高了几个数量级,但是,它们需要超网络或 RNN 控制器,并且主要关注小规模任务（例如 CIFAR)而不是大规模任务（例如

图片网）。

我们的工作与 One-Shot (Bender et al., 2018)和 DARTS (Liu et al., 2018c)关系最密切,两者都通过将 NAS 建模为单一训练来摆脱元控制器（或超网络)包含所有候选路径的过度参数化网络的过程。具体来说,One Shot 使用 DropPath (Zoph et al., 2018)训练过参数化网络,该网络以固定概率丢弃每条路径。然后,他们使用预训练的过度参数化网络来评估架构,这些架构是通过随机清零路径进行采样的。 DARTS 还为每条路径引入了实值架构参数,并通过标准梯度下降联合训练权重参数和架构参数。但是,它们存在 GPU 内存消耗大的问题,因此仍然需要使用代理任务。在这项工作中,我们通过路径二值化解决了这两种方法中的大内存问题。

另一个相关主题是网络修剪(Han et al., 2016) ,旨在通过去除无关紧要的神经元(Han et al., 2015)或通道(Liu et al., 2017)来提高神经网络的效率。
与这些工作类似,我们从一个过度参数化的网络开始,然后修剪冗余部分以获得优化的架构。不同之处在于,他们专注于层级剪枝,只修改一层的过滤器（或单元)数量,但不能改变网络的拓扑结构,而我们专注于通过路径级剪枝学习有效的网络架构。我们还允许修剪和增加层数。

## 3方法

我们首先描述了具有所有候选路径的过参数化网络的构造,然后介绍了我们如何利用二值化架构参数将训练过参数化网络的内存消耗降低到与常规训练相同的水平。我们提出了一种基于梯度的算法来训练这些二值化架构参数。最后,我们提出了两种技术来处理不可微目标（例如延迟） ,以便在目标硬件上专门化神经网络。

### 3.1超参数化网络的构建

将神经网络表示为 N (e, · · · , en) ,其中ei表示有向无环图 (DAG) 中的某条边。令 O = {oi}为 N 个候选原始操作的集合（例如卷积、池化、身份、零等）。为了构建在搜索空间中包含任何架构的过参数化网络,我们不是将每条边设置为确定的原始操作,而是将每条边设置为具有 N 条并行路径的混合操作（图2） ,表示为mO .因此,过参数化网络可以表示为 N (e = m1 en = mn O)。

哦， · · · ，

给定输入 x,混合运算mO的输出是基于其 N 条路径的输出定义的。在 One-Shot 中， mO(x) 是{oi(x)} 的总和,而在 DARTS 中， mO(x) 是 {oi(x)} 的加权和,其中

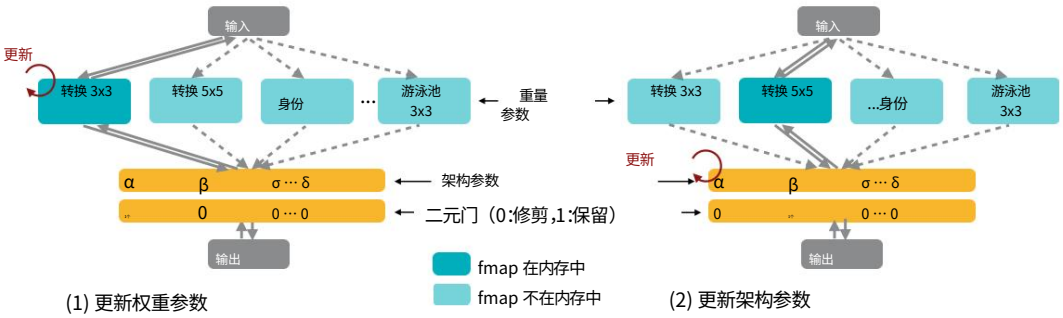


图 2:学习权重参数和二值化架构参数。

通过将 softmax 应用于 N 个实值架构参数{αi} 来计算权重：

$$m_{\text{One-Shot}}(x) = \sum_{i=1}^{N} o_i(x), \qquad m_{\text{DARTS}}(x) = \sum_{i=1}^{N} p_i o_i(x) = \sum_{i=1}^{N} \frac{\exp(\alpha_i)}{\sum_j \exp(\alpha_j)} o_i(x)。 \tag{1}$$

如方程式所示。 (1),计算所有N条路径的输出特征图并存储在内存中,而训练紧凑模型只涉及一条路径。因此,与训练紧凑型模型相比,One-Shot 和 DARTS 大致需要 N 倍的 GPU 内存和 GPU 小时数。在大规模数据集,这很容易超过具有大设计空间的硬件的内存限制。在下一节中,我们基于路径二值化的思想来解决这个内存问题。

直接测量：又贵又慢

延迟建模:廉价、快速且可区分

3.2学习二值化路径

为了减少内存占用,我们在训练过度参数化网络时只保留一条路径。
与Courbariaux 等人不同。 (2015)将单个权重二值化,我们将整个路径二值化。我们引入 N 个实值架构参数{αi} ,然后将实值路径权重转换为二进制门：

$$g = \text{二值化}(p_1, \cdots, p_N) = \begin{cases} [1, 0, \cdots, 0] & \text{概率为} p_1, \\ \cdots & \\ [0, 0, \cdots, 1] & \text{概率为} p_N。 \end{cases} \tag{2}$$

$$E[\text{延迟}] = \sum_i p_i \cdot F(\text{op}_i)$$

F(conv 3x3)
F(conv 5x5)
F(identity)
……
F（泳池 3x3）

α β ··· ζ
可学习模块i - 1
转换 3x3 转换 5x5 身份 …… 游泳池 3x3

基于二元门g,混合运算的输出为：

$$m_{\text{二进制}}(x) = \sum_{i=1}^{N} g_i o_i(x) = \begin{cases} o_1(x) & \text{概率为} p_1 \\ \cdots & \\ o_N(x) & \text{概率为} p_N。 \end{cases} \tag{3}$$

$$E[\text{延迟}] = X \cdot E[\text{潜伏期}]$$

$$\text{损失} = \text{LossCE} + \lambda_1 \|w\|_2^2 + \lambda_2 E$$

可学习块
……

如方程式所示。 (3)和图2,通过使用二进制门而不是实值路径权重(Liu et al., 2018c),在运行时只有一条激活路径在内存中处于活动状态,并且训练过度的内存需求参数化网络因此减少到与训练紧凑模型相同的水平。这不仅仅是一个数量级的内存节省。

3.2.1训练二值化架构参数

图2说明了过参数化网络中权重参数和二值化架构参数的训练过程。在训练权重参数时,我们首先冻结架构参数并根据等式随机采样二元门。 (2)对于每批输入数据。然后通过训练集上的标准梯度下降更新活动路径的权重参数（图2左）。在训练架构参数时,权重参数被冻结,然后我们重置二进制门并更新验证集上的架构参数（图2右）。这两个更新步骤以替代方式执行。一旦架构参数的训练完成,我们就可以通过修剪冗余路径来导出紧凑的架构。在这项工作中,我们只需选择路径权重最高的路径。

与权重参数不同,架构参数不直接参与计算图,因此无法使用标准梯度下降法进行更新。在本节中,我们介绍了一种基于梯度的方法来学习架构参数。

在 BinaryConnect (Courbariaux et al., 2015) 中,实值权重使用其对应的二进制门的梯度进行更新。在我们的例子中,类似地,梯度 wrt 架构参数可以使用 ∂L/∂gi代替 ∂L/∂pi来近似估计:

$$\frac{\partial L}{\partial \alpha_i} = \sum_{j=1}^{N} \frac{\partial L}{\partial p_j}\frac{\partial p_j}{\partial \alpha_i} \approx \sum_{j=1}^{N} \frac{\partial L}{\partial g_j}\frac{\partial p_j}{\partial \alpha_i} = \sum_{j=1}^{N} \frac{\partial L}{\partial g_j}\frac{\partial \frac{\exp(\alpha_j)}{\sum_k \exp(\alpha_k)}}{\partial \alpha_i} = \sum_{j=1}^{N} \frac{\partial L}{\partial g_j} p_j (\delta_{ij} - p_i), \tag{4}$$

其中,如果 i = j,则δij = 1,如果 i = j,则δij = 0。由于二元门 g 参与了计算图,如等式 1 所示。 (3)、∂L/∂gj可以通过反向传播计算。然而,计算∂L/∂gj需要计算并存储oj (x)。因此,直接使用方程式。 (4)与训练紧凑模型相比,更新架构参数也需要大约 N 倍的 GPU 内存。

为了解决这个问题,我们考虑将从 N 个候选路径中选择一条路径的任务分解为多个二元选择任务。直觉是,如果一条路径是特定位置的最佳选择,那么与任何其他路径相比,它应该是更好的选择。 2

按照这个想法,在架构参数的更新步骤中,我们首先根据多项分布(p1, · · · , pN ) 对两条路径进行采样,并屏蔽所有其他路径,就好像它们不存在一样。因此,候选者的数量暂时从 N 减少到 2,同时路径权重的更新不受影响。接下来,我们计算梯度并更新这两条采样路径的架构参数。最后,我们通过计算出将 softmax 应用于架构参数来计算路径权重,我们需要通过乘以一个比率来重新缩放这两个更新的架构参数的值,以保持未采样路径的路径权重不变。因此,在每个更新步骤中,一个采样路径被增强（路径权重增加）,另一个采样路径被衰减（路径权重减小）,而所有其他路径保持不变。这样,无论N的值如何,架构参数的每个更新步骤都只涉及两条路径,从而将内存需求降低到与训练紧凑模型相同的水平。

### 3.3处理不可微分的硬件指标

在为硬件设计高效的神经网络架构时,除了准确性之外,延迟（不是 FLOPs)是另一个非常重要的目标。不幸的是,与可以使用损失函数的梯度优化的准确性不同,延迟是不可微分的。在本节中,我们提出了两种算法来处理不可微分的目标。

#### 3.3.1使延迟可微分

为了使延迟可区分,我们将网络的延迟建模为 的连续函数。考虑具有候选集{oj}的混合操作,每个oj是与路径权重pj关联的神经网络[2] 维度,表示选择oj的概率。

因此,我们将混合操作（即可学习块)的预期延迟表示为:

$$E[\text{潜伏期}] = \sum_j \text{latency}_i(\text{可学习块}_j) \times F(o_j), p_j \tag{5}$$

其中 E[latencyi ] 是 i tion 模型的预期延迟,F(o ) 是 o 参数的预测延迟,因此可以给出:∂E[latencyi ] / ∂pi

$$F(o_j) = F(o_j \text{ 的梯度})。$$

对于具有一系列混合操作的整个网络（图3左）,由于这些操作在推理过程中按顺序执行,因此网络的预期延迟可以用这些混合操作的预期延迟之和表示:

$$E[\text{延迟}] = \sum_l E[\text{潜伏期}_l], \tag{6}$$

[2] 在附录D 中,我们提供了另一种不需要近似的解决方案。
3 延迟预测模型的详细信息在附录B中提供。

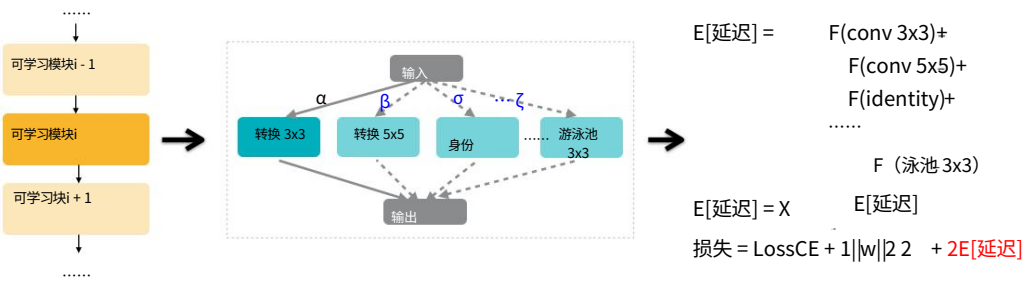


图 3:通过引入延迟正则化损失使延迟可区分。

我们通过乘以比例因子λ2(> 0) 将网络的预期延迟合并到正常损失函数中,该比例因子控制准确性和延迟之间的权衡。最终的损失函数给出为（也如图3右所示）

$$\text{Loss} = \text{LossCE} + \lambda_1 ||w||_2^2 + \lambda_2 E[\text{latency}], \text{其中 LossCE} \tag{7}$$

表示交叉熵损失， λ1||w||2 2是权重衰减项。

3.3.2 基于强化的方法

作为 BinaryConnect 的替代方案,我们也可以利用 REINFORCE 来训练二值化权重。
考虑一个具有二值化参数 α 的网络,更新二值化参数的目标是找到最大化某个奖励的最佳二元门 g,表示为 R(·)。这里我们假设网络只有一种混合操作以便于说明。因此,根据 REINFORCE (Williams, 1992),我们对二值化参数进行了以下更新：

$$J(\alpha) = E_{g \sim \alpha}[R(g)] = \sum_i p_i R(N(e = o_i)),$$

$$\nabla_\alpha J(\alpha) = \sum_i R(N(e = o_i)) \nabla_\alpha p_i = \sum_i R(N(e = o_i)) p_i \nabla_\alpha \log(p_i),$$

$$= E_{g \sim \alpha}[R(g) \nabla_\alpha \log(p(g))] \approx \frac{1}{M} \sum_{i=1}^{M} R(g_i) \nabla_\alpha \log(p(g_i)), \tag{8}$$

其中 $g_i$表示第 i个采样二进制门,$p(g_i)$ 表示采样 $g_i$ 的概率。我同意 Eq。 (2)并且$g_i$是根据二进制门 g 的紧凑网络。由于等式。 (8)不要求 R(g)对 g 是可微的,因此它可以处理不可微的目标。一个有趣的观察是方程式。 (8)与标准 NAS (Zoph & Le, 2017)具有相似的形式,但它不是顺序决策过程,并且在我们的案例中没有使用 RNN 元控制器。

此外,由于基于梯度的更新和基于 REINFORCE 的更新本质上是对相同二值化架构参数的两种不同更新规则,因此可以将它们组合起来形成新的架构参数更新规则。

## 4实验和结果

我们在图像分类任务的两个基准数据集（CIFAR-10 和 ImageNet)上证明了我们提出的方法的有效性。与之前的 NAS 作品（Zoph 等人， 2018； Liu 等人， 2018c)不同,它首先在小规模设置（例如更少的块)下学习 CIFAR-10 上的 CNN 块,然后将学习到的块传输到 ImageNet 或 CIFAR-10在大规模设置下,通过重复堆叠,我们直接学习目标任务（CIFAR-10 或 ImageNet)和目标硬件（GPU、CPU 和手机)的架构,同时允许指定每个块。

### 4.1 CIFAR -10 上的实验

建筑空间。对于 CIFAR-10 实验,我们使用Cai 等人介绍的树结构空间。 (2018b)以 PyramidNet (Han et al., 2017)作为主干4 。具体来说，

4候选集中的操作列表在附录中提供。

| 模型 | 参数 | 测试误差（%） |
|---|---|---|
| DenseNet-BC （Huang 等人， 2017 年） | 25.6M | 3.46 |
| PyramidNet (Han et al., 2017) | 26.0M | 3.31 |
| Shake-Shake + c/o (DeVries & Taylor, 2017) | 26.2M | 2.56 |
| PyramidNet + SD (Yamada et al., 2018) | 26.0M | 2.31 |
| ENAS + c/o （Pham 等人， 2018 年） | 4.6M | 2.89 |
| DARTS + c/o (Liu et al., 2018c) | 3.4M | 2.83 |
| NASNet-A + c/o （Zoph 等人， 2018 年） | 27.6M | 2.40 |
| PathLevel EAS + c/o （Cai 等人， 2018b） | 14.3M | 2.30 |
| AmoebaNet-B + c/o （Real 等人， 2018 年） | 34.9M | 2.13 |
| Proxyless-R + c/o（我们的） | 5.8M | 2.30 |
| Proxyless-G + c/o（我们的） | 5.7M | 2.08 |

表 1:ProxylessNAS 在 CIFAR-10 上实现了最先进的性能。

我们将 PyramidNet 残差块中的所有 3 × 3 卷积层替换为树结构单元,每个单元的深度为 3,每个节点（叶节点除外)的分支数设置为 2。有关树结构空间的更多详细信息,请参阅原始论文（Cai 等人， 2018b）。此外,我们使用两个超参数来控制该架构空间中网络的深度和宽度,即 B 和 F,分别表示每个阶段（共 3 个阶段)的块数和最终块的输出通道数.

训练细节。我们从训练集中随机抽取 5,000 张图像作为学习架构参数的验证集,这些参数使用 Adam 优化器进行更新,基于梯度的算法（第3.2.1 节)的初始学习率为 0.006,基于REINFORCE 的初始学习率为 0.01算法（第3.3.2 节）。在下面的讨论中,我们将这两种算法分别称为Proxyless-G （梯度)和Proxyless-R （REINFORCE）。

过参数化网络的训练过程完成后,根据架构参数导出紧凑网络,如第3.2.1 节所述。接下来,我们使用相同的训练设置训练紧凑网络,只是训练时期的数量从 200 增加到 300。此外,当采用 DropPath 正则化（Zoph 等人， 2018 年； Huang 等人， 2016 年)时,我们进一步将训练时期的数量增加到 600 （Zoph 等人， 2018）。

结果。我们应用所提出的方法在 B = 18 和 F = 400 的树结构空间中学习架构。由于我们不重复单元并且每个单元有 12 个可学习边,因此总共需要 12 × 18 × 3 = 648 个决策来完全确定架构。

我们提出的方法和其他最先进的架构在 CIFAR 10 上的测试错误率结果总结在表1 中,其中“c/o”表示使用 Cutout （DeVries & Taylor， 2017）。
与这些最先进的架构相比,我们提出的方法不仅可以实现更低的测试错误率,还可以实现更好的参数效率。具体来说,Proxyless-G 达到了 2.08% 的测试错误率,略好于 AmoebaNet-B (Real et al., 2018) （之前 CIFAR-10 上的最佳架构）。值得注意的是,AmoebaNet-B 使用 34.9M 参数,而我们的模型仅使用 5.7M 参数,比 AmoebaNet-B 少 6 倍。此外,与同样探索树结构空间的PathLevel EAS (Cai et al., 2018b)相比,Proxyless-G 和 Proxyless-R 都以减少一半的参数实现了相似或更低的测试错误率结果。我们的 ProxylessNAS 的强大实证结果证明了直接探索大型架构空间而不是重复堆叠同一块的好处。

### 4.2 IMAGENET上的实验

对于 ImageNet 实验,我们专注于学习高效的 CNN 架构（Iandola et al., 2016; Howard et al., 2017; Sandler et al., 2018; Zhu et al., 2018） ,它们不仅具有高精度而且低延迟在特定的硬件平台上。因此,它是一个多目标 NAS 任务(Hsu et al., 2018; Dong et al., 2018; Elsken et al., 2018a; He et al., 2018; Wang et al., 2018; Tan et al. , 2018),其中一个目标是不可微分的（即延迟）。我们在实验中使用了三种不同的硬件平台,包括手机、GPU 和 CPU。 GPU 延迟是在批大小为 8 的 V100 GPU 上测量的（单批会使 GPU 严重利用不足）。

CPU 延迟是在具有两个 2.40GHz Intel(R) Xeon(R) 的服务器上以批量大小 1 测量的

| 模型 | Top-1 | Top-5 | 移动硬件 | 无 | 无搜索成本 | 延迟感知代理 | 重复（GPU | 小时） | 89.5 |
|---|---|---|---|---|---|---|---|---|---|
| | | | 113ms | 91.0 | 75ms | 91.3 | 183ms | 92.0 | 190ms | 91.8 | 76ms | 91.8 | 79ms |
| MobileNetV1 [16] | 70.6 | 90.3 | 91.7 | 79ms | 92.2 | 78ms | - | - | | 手动的 |
| MobileNetV2 [30] | 72.0 | NASNet-A [38] | 74.0 | | | - | - | | | 手动 | 48、 |
| AmoebaNet-A [29] | 74.5 | MnasNet [31] | 74.0 | | | | | | | 000 | 75、 |
| MnasNet（我们的实现） | 74.0 | Proxyless-G（移动） | 71.8 | Proxyless-G+ | | | | | | 600 | 40、 |
| Proxyless-LL | 74.2 | Proxyless -R（手机） | 74.6 | | | | | | | 000 | 40、 |
| | | | | | | | | | | 000 | 200 |
| | | | 83毫秒 | ✓ | | | | ✓ | | 200 |
| | | | | ✓ | | | | ✓ | | |
| | | | | ✓✓ | | ✓✓✓ | | ✓ | | 200 |

表 2:ProxylessNAS 在 ImageNet 上实现了最先进的准确率 (%)（在移动延迟限制≤ 80 毫秒的情况下）,GPU 小时内的搜索成本降低了 200 倍。 “LL”表示延迟正则化损失。附录C中提供了 MnasNet 搜索成本的详细信息。

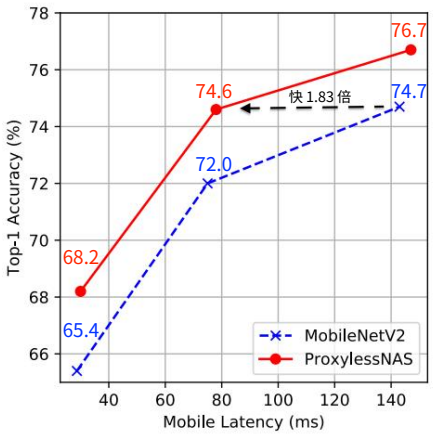


图 4:ProxylessNAS 在各种延迟设置下始终优于 MobileNetV2。

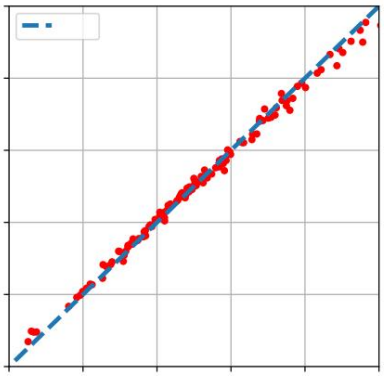

图 5:我们的移动延迟模型接近 y = x。延迟 RMSE 为 0.75 毫秒。

中央处理器 E5-2640 v4.移动延迟是在批量大小为 1 的 Google Pixel 1 手机上测量的。对于 Proxyless-R,我们使用 ACC(m) × [LAT(m)/T] w作为优化目标,其中 ACC(m) 表示模型m的精度,LAT(m)表示的延迟。T是目标延迟,w是控制精度和延迟之间权衡的超参数。

此外,在手机上,我们在架构搜索期间使用延迟预测模型（附录 B）。如图5 所示,我们在测试集上观察到预测延迟与实际测量延迟之间存在很强的相关性,这表明延迟预测模型可用于替换昂贵的移动农场基础设施（Tan 等人， 2018 年）,成本极低错误介绍。

建筑空间。我们使用 MobileNetV2 （Sandler 等人， 2018)作为构建架构空间的主干。具体来说,我们允许一组具有不同内核大小 {3, 5, 7} 和扩展比率 {3, 6} 的 MBConv 层,而不是重复相同的移动反向瓶颈卷积 (MBConv)。为了实现宽度和深度之间的直接权衡,我们启动了一个更深的超参数化网络,并允许通过将零操作添加到其混合操作的候选集来跳过具有残差连接的块。这样,在延迟预算有限的情况下,网络可以通过跳过更多块和使用更大的 MBConv 层来选择更浅和更宽,或者通过保留更多块和使用更小的 MBConv 层来选择更深和更薄。

训练细节。在架构搜索期间,我们从训练集中随机抽取 50,000 张图像作为验证集。更新架构参数的设置与 CIFAR-10 实验相同,只是初始学习率为 0.001。过参数化网络在剩余的训练图像上进行训练,批量大小为 256。

| 模型 | Top-1 | Top-5 | GPU 延迟 |
|---|---|---|---|
| MobileNetV2 (Sandler et al., 2018) | 72.0 | 91.0 | 6.1ms |
| ShuffleNetV2 (1.5) (Ma et al., 2018) | 72.6 | [illegible] | [illegible] |
| DARTS （Liu 等人， 2018c） | 73.1 | [illegible] | [illegible] |
| MnasNet （Tan 等人， 2018b） | 74.8 | 91.4 | 38.3ms |
| 无代理（GPU） | | 91.3 | |
| | | 91.0 | - |
| | | 91.8 | 6.1毫秒 |
| | | 92.5 | 5.1毫秒 |

表 3:ImageNet 上的 ImageNet 精度 (%) 和 GPU 延迟 (Tesla V100)。

ImageNet 分类结果。我们首先应用我们的 ProxylessNAS 在手机上学习专门的 CNN 模型。汇总结果如表2 所示。与 MobileNetV2 相比,我们的模型将 top-1 准确率提高了 2.6%,同时在手机上保持了类似的延迟。此外,通过使用乘数重新调整网络宽度（San dler 等人， 2018 年； Tan 等人， 2018 年）,如图4所示,我们的模型在所有延迟设置下始终以显着优势优于 MobileNetV2。具体来说,为了达到相同水平的 top-1 精度性能（即大约 74.6%）， MobileNetV2 有 143ms 的延迟,而我们的模型只需要 78ms（快 1.83 倍）。与 MnasNet (Tan et al., 2018) 相比,我们的模型可以实现高出 0.6% 的 top-1 准确率,同时移动延迟略低。更重要的是,我们的资源效率更高:GPU 小时比 MnasNet 少 200 倍（表2）。

此外,我们还观察到,如果不是因为延迟正则化损失,Proxyless-G 没有动机选择计算成本低的操作。其生成的架构最初在 Pixel 1 上具有 158 毫秒的延迟。在使用乘数重新缩放网络后,其延迟减少到 83 毫秒。

然而,该模型在 ImageNet 上只能达到 71.8% 的 top-1 精度,比带有延迟正则化损失的 Proxyless-G 给出的结果低 2.4%。因此,我们得出结论,在学习高效神经网络时,必须将延迟作为直接目标。

除了手机,我们还应用我们的 ProxylessNAS 在 GPU 和 CPU 上学习专门的 CNN 模型。表3报告了 GPU 上的结果,我们发现与人工设计和自动搜索的架构相比,我们的 ProxylessNAS 仍然可以实现卓越的性能。具体来说,与 MobileNetV2 和 MnasNet 相比,我们的模型分别将 top-1 准确率提高了 3.1% 和 1.1%,同时速度提高了 1.2 倍。表4显示了我们在三个不同平台上搜索模型的总结结果。一个有趣的观察是,针对 GPU 优化的模型在 CPU 和手机上运行速度并不快,反之亦然。因此,有必要针对不同的硬件架构学习专门的神经网络,以在不同的硬件上实现最佳效率。

针对不同硬件的专用模型。图6展示了我们在三个硬件平台上搜索的 CNN 模型的详细架构:GPU/CPU/Mobile。我们注意到,当针对不同的平台时,架构表现出不同的偏好：(i)GPU 模型更浅和更宽,特别是在特征图具有更高分辨率的早期阶段； (ii) GPU 模型更喜欢大型 MBConv 操作（例如 7 × 7 MBConv6）,而 CPU 模型会选择较小的 MBConv 操作。这是因为 GPU 的并行度比 CPU 高得多,因此它可以利用大型 MBConv 操作。另一个有趣的观察结果是,我们在所有平台上搜索的模型更喜欢在特征图被下采样的每个阶段的第一个块中使用更大的 MBConv 操作。我们认为这可能是因为更大的 MBConv 操作有利于网络在下采样时保留更多信息。值得注意的是,这种模式无法在以前的 NAS 方法中捕获,因为它们迫使块共享相同的结构（Zoph et al., 2018; Liu et al., 2018a）。

## 5结论

我们引入了 ProxylessNAS,它可以在没有任何代理的情况下直接在目标任务和目标硬件上学习神经网络架构。我们还使用路径二值化将 NAS 的搜索成本（GPU 小时和 GPU 内存)降低到与正常训练相同的水平。受益于直接搜索,我们在 CIFAR-10 和 ImageNet 上取得了很好的实证结果。此外,

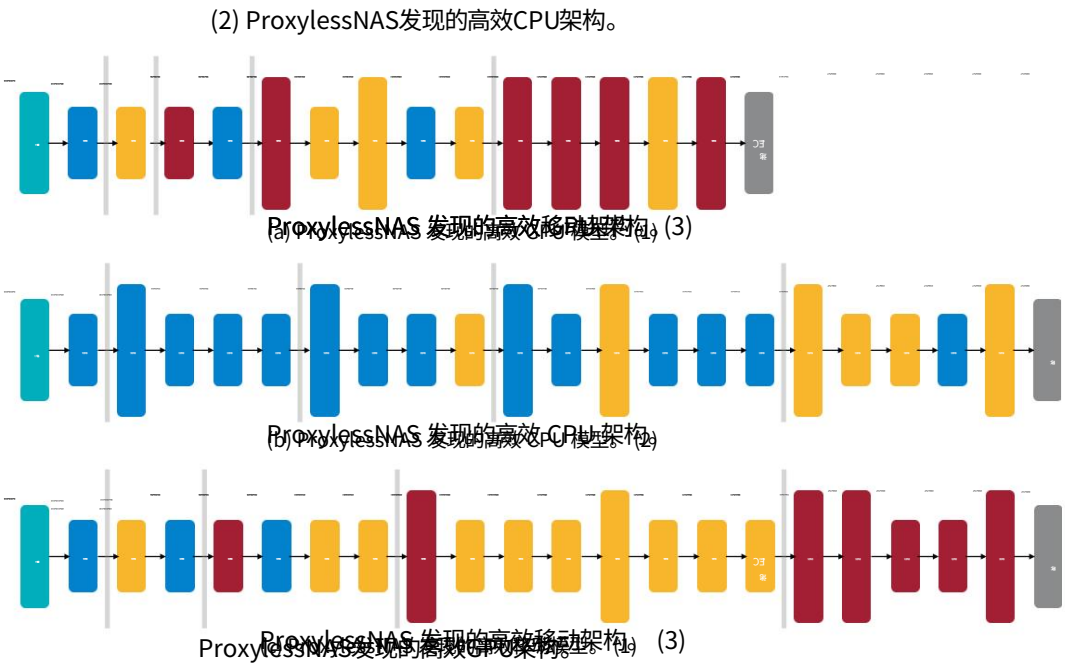


图 6:针对不同硬件优化的高效模型。“MBConv3”和“MBConv6”分别表示移动反向瓶颈卷积层,扩展率为 3 和 6。展望:GPU 更喜欢带有早期池化的浅而宽的模型; CPU 更喜欢带有延迟池的深度和狭窄模型。池化层更喜欢大而宽的内核。早期层更喜欢小内核。后期层更喜欢大内核。

(2) ProxylessNAS发现的高效CPU架构。

| 模型 | Top-1 (%) | GPU 延迟 | CPU 延迟 | 移动延迟 |
|---|---|---|---|---|
| 无代理（GPU） | 75.1 | 5.1毫秒 | 204.9毫秒 | 124 毫秒 |
| 无代理（CPU） | 75.3 | 7.4毫秒 | 138.7 毫秒 | 116 毫秒 |
| 无代理（移动） | 74.6 | 7.2毫秒 | 164.1毫秒 | 78毫秒 |

表 4:硬件更喜欢专用模型。针对 GPU 优化的模型在 CPU 和手机上运行不快,反之亦然。 ProxylessNAS 提供了一种有效的解决方案来搜索ProxylessNAS 发现的专用 (3) 高效 GPU 架构。目标硬件架构的神经网络架构,同时与最先进的技术相比,搜索成本降低了 200 倍（Zoph & Le, 2017; Tan et al., 2018）。

我们允许通过将测量的硬件延迟直接纳入优化目标来为不同平台专门化网络架构。我们比较了 CPU/GPU/移动设备上的优化模型,并提高了对针对不同硬件架构的专用神经网络架构需求的认识。

## 致谢

我们感谢 MIT Quest for Intelligence、MIT-IBM Watson AI lab、SenseTime、Xilinx、Snap Research 对这项工作的支持。我们也感谢 AWS Cloud Credits for Research Program 为我们提供的云计算资源。

## 参考

Gabriel Bender、Pieter-Jan Kindermans、Barret Zoph、Vijay Vasudevan 和 Quoc Le。理解进行和简化一次性架构搜索。在 ICML,2018 年。

Andrew Brock、Theodore Lim、James M Ritchie 和 Nick Weston。 Smash:一次性模型存档通过超网络进行结构搜索。在 ICLR,2018 年。

韩才、陈天耀、张维南、余勇和王军。高效的架构搜索网络改造。在 AAAI 中,2018a。

蔡寒、杨家成、张维南、韩松、于勇。路径级网络转换
用于高效的架构搜索。在 ICML 中,2018b。

Matthieu Courbariaux、Yoshua Bengio 和 Jean-Pierre David。 Binaryconnect:训练深度神经网络
传播期间具有二进制权重的网络。在 NIPS 中,2015 年。

特伦斯·德弗里斯和格雷厄姆·W·泰勒。改进卷积神经网络的正则化
与镂空。 arXiv 预印本 arXiv:1708.04552, 2017。

Jin-Dong Dong、An-Chieh Cheng、Da-Cheng Juan、Wei Wei 和 Min Sun。 Dpp-net:设备感知
渐进式搜索帕累托最优神经结构。在 ECCV,2018 年。

Thomas Elsken、Jan-Hendrik Metzen 和 Frank Hutter。简单高效的卷积神经网络架构搜索。 arXiv 预印本
arXiv:1711.04528, 2017。

Thomas Elsken、Jan Hendrik Metzen 和 Frank Hutter。 CNN 的多目标架构搜索。 arXiv 预印本 arXiv:1804.09081,
2018a。

Thomas Elsken、Jan Hendrik Metzen 和 Frank Hutter。神经架构搜索:一项调查。 arXiv
预印本 arXiv:1808.05377, 2018b。

Dongyoon Han、Jiwhan Kim 和 Junmo Kim。深金字塔残差网络。在 CVPR,2017 年。

Song Han、Jeff Pool、John Tran 和 William Dally。学习权重和连接
高效的神经网络。在 NIPS 中,2015 年。

Song Han、Huizi Mao 和 William J Dally。深度压缩:通过修剪、训练量化和霍夫曼编码压缩深度神经网络。在 ICLR,
2016 年。

何开明、张翔宇、任少卿和孙健。用于图像识别的深度残差学习。在 CVPR,2016 年。

Yihui He、Ji Lin、Zhijian Liu、Hanrui Wang、Li-Jia Li 和 Song Han。 Amc:模型的 Automl
移动设备上的压缩和加速。在 ECCV,2018 年。

Andrew G Howard、Menglong Zhu、Bo Chen、Dmitry Kalenichenko、Weijun Wang、Tobias Weyand、Marco
Andreetto 和 Hartwig Adam。 Mobilenets:用于移动视觉应用的高效卷积神经网络。 arXiv 预印本
arXiv:1704.04861, 2017。

Chi-Hung Hsu、Shu-Huan Chang、Da-Cheng Juan、Jia-Yu Pan、Yu-Ting Chen、Wei Wei 和 Shih Jieh Chang。
Monas:使用强化学习的多目标神经架构搜索。 arXiv 预印本 arXiv:1806.10332, 2018。

Gao Huang、Yu Sun、Zhuang Liu、Daniel Sedra 和 Kilian Q Weinberger。深层网络
随机深度。在 ECCV,2016 年。

高煌、刘庄、劳伦斯范德马腾和 Kilian Q Weinberger。紧密相连
卷积网络。在 CVPR,2017 年。

Forrest N Iandola、Song Han、Matthew W Moskewicz、Khalid Ashraf、William J Dally 和 Kurt Keutzer。
Squeezenet:Alexnet 级精度,参数减少 50 倍,模型大小为 0.5 mb。 arXiv 预印本 arXiv:1602.07360, 2016。

Purushotham Kamath、Abhishek Singh 和 Debo Dutta。神经架构构建使用
信封网。 arXiv 预印本 arXiv:1803.06744, 2018。

Chenxi Liu、Barret Zoph、Jonathon Shlens、Wei Hua、Li-Jia Li、Li Fei-Fei、Alan Yuille、Jonathan Huang 和 Kevin
Murphy。渐进式神经架构搜索。在 ECCV 中,2018a。

Hanxiao Liu、Karen Simonyan、Oriol Vinyals、Chrisantha Fernando 和 Koray Kavukcuoglu。你好
高效架构搜索的分层表示。在 ICLR 中,2018b。

Hanxiao Liu、Karen Simonyan 和 Yiming Yang。 Darts:可区分的架构搜索。 arXiv
预印本 arXiv:1806.09055, 2018c。

刘庄、李建国、沉志强、黄高、闫守梦和张长水。学习
通过网络瘦身来构建高效的卷积网络。在 ICCV,2017 年。

Renqian Luo, Fei Tian, Tao Qin, and Tie-Yan Liu.神经结构优化。 arXiv 预印本
arXiv:1808.07233, 2018.

马宁宁、张翔宇、郑海涛和孙健。 Shufflenet v2:实用指南
高效的CNN架构设计。在 ECCV,2018 年。

Hieu Pham、Melody Y Guan、Barret Zoph、Quoc V Le 和 Jeff Dean。通过参数共享进行高效的神经结构搜索。在
ICML,2018 年。

Esteban Real、Alok Aggarwal、Yanping Huang 和 Quoc V Le。图像的正则化进化
分类器架构搜索。 arXiv 预印本 arXiv:1802.01548, 2018。

Mark Sandler、Andrew Howard、Menglong Zhu、Andrey Zhmoginov 和 Liang-Chieh Chen。莫
bilenetv2:倒置残差和线性瓶颈。在 CVPR,2018 年。

Mingxing Tan、Bo Chen、Ruoming Pang、Vijay Vasudevan 和 Quoc V Le。 Mnasnet:移动平台感知神经架构搜
索。 arXiv 预印本 arXiv:1807.11626, 2018。

王宽、刘志坚、林玉君、吉林、韩松。 Haq:硬件感知自动量化。 arXiv,2018 年。

罗纳德·J·威廉姆斯。用于连接强化学习的简单统计梯度跟踪算法。在强化学习中。 1992.

Yoshihiro Yamada、Masakazu Iwamura 和 Koichi Kise。 Shakedrop 正则化。 arXiv 预印本
arXiv:1802.02375, 2018.

Zhao Zhong、Junjie Yan、Wei Wu、Jing Shao 和 Cheng-Lin Liu。实用的块式神经网络
网络架构生成。在 CVPR,2018 年。

朱力耕、邓睿智、Michael Maire、邓志伟、Greg Mori 和谭平。稀疏聚集的卷积网络。在欧洲计算机视觉会议 (ECCV) 的会
议记录中,第 186–201 页,2018 年。

Barret Zoph 和 Quoc V Le。具有强化学习的神经结构搜索。在 ICLR,2017 年。

Barret Zoph、Vijay Vasudevan、Jonathon Shlens 和 Quoc V Le。学习用于可扩展图像识别的可迁移架构。在 CVPR,
2018 年。

## A CIFAR -10 上使用的候选操作列表

我们在 CIFAR-10 实验中采用了以下 7 个操作：

· 3 × 3 dilated depthwise-separable convolution
· Identity · 3 × 3 depthwise-separable convolution · 5 × 5 depthwise-separable convolution · 7 × 7 depthwise-separable convolution · 3 × 3 average pooling · 3 × 3 max pooling

## B移动延迟预测

测量设备上的延迟是准确的,但对于可扩展的神经架构搜索来说并不理想。
有两个原因：(i)慢。正如 TensorFlow-Lite 中所建议的,我们需要平均数百次运行才能产生精确的测量结果,大约需要 20 秒。这比单个前向/后向执行要慢得多。 (ii) 昂贵。需要大量的移动设备和软件工程工作来构建自动管道以收集来自移动农场的延迟。我们没有直接测量,而是建立了一个模型来估计延迟。我们只需要一部手机而不是一堆手机,它的延迟 RMSE 仅为 0.75 毫秒。我们使用延迟模型进行搜索,并使用测量的延迟来报告最终模型的延迟。

我们从我们的候选空间中采样了 5k 架构,其中 4k 架构用于构建延迟模型,其余用于测试。我们使用 TensorFlow-Lite 测量了 Google Pixel 1 手机上的延迟。这些特征包括（i)算子的类型（ii)输入和输出特征图大小（iii)其他属性,如内核大小、卷积步长和扩展率。

## C MNASNET搜索成本的详细信息

Mnas （Tan 等人， 2018 年)在 ImageNet 上训练了 8,000 个移动大小的模型,每个模型都训练了 5 个 epoch,用于学习架构。如果像我们的实验那样在 V100 GPU 上训练这些模型,则搜索成本约为 40,000 GPU 小时。

## 基于梯度的算法的实现

基于梯度的算法（见等式（4）)的一个简单实现是在前向步骤中计算和存储oj (x),以便稍后在后向步骤中计算 ∂L/∂gj： ∂L/∂gj = reduce sum (∇yL ◦ oj (x)),

(9)

其中∇yL表示混合操作y输出的梯度，“◦”表示元素乘积，“reduce sum(·)”表示所有元素的总和。

请注意,当 j 路径未激活（即不参与计算 y)时， oj (x) 仅用于计算 ∂L/∂gj。所以我们不需要实际分配GPU内存来存储 oj (x)。相反,我们可以在反向步骤中得到∇yL后计算oj (x) ,使用oj (x) 计算 ∂L/∂gj,如下式。 (9)、然后释放占用的GPU内存。通过这种方式,无需3.2.1节中讨论的近似,我们可以将 GPU 内存成本降低到与训练紧凑模型相同的水平。