Headline

大家好：

这是**2018年度第4篇Arxiv Weekly**。

本文是 模型压缩、**NAS** 方向的文章。

Highlight

> **Song Han**组新工作，利用网络自动搜索技术进行网络压缩，并取得超过手动压缩的性能。

Information

Title

AMC: AutoML for Model Compression and Acceleration on Mobile Devices

Link

https://arxiv.org/pdf/1802.03494.pdf

Source

- 麻省理工大学（Massachusetts Institute of Technology）
- 西安交大（Xian Jiaotong University）
- 谷歌（Google）

Introduction

模型压缩已经成为神经网络移动化技术的核心之一。在有限的计算资源和能量条件下如何实现模型的部署，是神经网络落地的关键问题。

CNN压缩技术的出现给这个问题提供了一个有力的解决方案。然而传统的模型压缩是纯手动的，基本是基于一些启发式的规则（heuristic rule-based compression）探索模型储存、效率和性能的最佳平衡。 手动的启发式平衡点，显然是次优化的，而且这样的压缩十分耗时，需要有经验的工程师和大量的时间和算力。

本文中提出了自动化模型压缩策略（AutoML for Model Compression, AMC）, 基于强化学习来进行模型压缩。实验结果表明，相比于启发式规则指导的模型压缩，AMC能够获得更高的压缩率的同时保留更多的性能。

以FLOPs为指标，用AMC指导VGG的4倍压缩相比于手动压缩涨点2.7%。在MobileNet上利用AMC进一步压缩网络，在安卓手机平台和Titan XP GPU上分别实现了ImageNet Inference中1.81$\times$ 和1.43$\times$的加速，top1 acc仅仅降低了0.1%。

Keys

1. 先验pruning技巧

一般来说，启发式的压缩会考虑如下的因素：

- 尽量降低第一个layer的pruning rate，因为第一个layer往往参数量不大，而且其抽象的底层特征是否准确会影响后续的所有layer的性能。
- 尽量多在FC layer做压缩，因为FC layer一般非常臃肿，占据了极大的储存和计算资源，并且高层的特征重复度高。
- 尽量找准每个layer的敏感点，pruning rate不能超过这个阈值。

但是显然，这三条原则从最优性上不能得证，从可操作性上需要大量的实验去调整各个layer的阈值，最糟糕的是只要模型改变所有的努力都需要重来一次，先验成果推广困难。

2. 问题分类

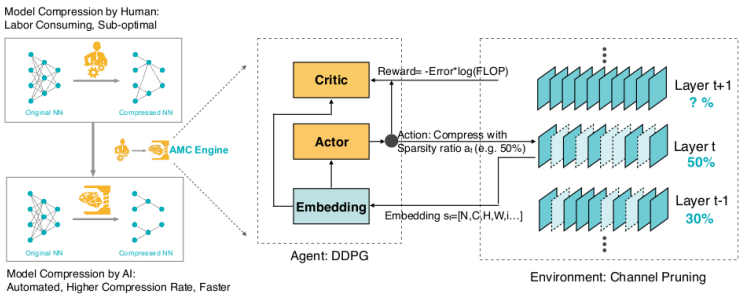
我们知道，传统的pruning研究分成两个大类：

- *Fine-grained pruning* 旨在独立地考察和剪枝weight tensor中的每个elements。由于操作空间大，可以几乎无损地实现较大比例的压缩，但是需要配合EIE一类的针对性硬件结构来保证inference的效率。
- *Coarse-grained pruning* 粗粒度pruning操作针对weight tensor中的某一块操作，例如一行、一列、一个channel。由于结构整饬，虽然压缩比例较小，但是inference的效率不受什么影响，能够在通用硬件上实现较好的性能。

本文中研究的是Coarse-grained pruning中channel层面的压缩，但是同样的方法可以用在其他类型的pruning当中。

3. AMC的pipeline

文章提出的AMC结构如下图所示：



可以看到整体上是标准的RL结构，而且能大体看出是Actor-Critic结构，具体的大家可以参考莫烦大佬的讲解和代码。实际上本文是使用了Actor-Critic结构的最新变体DDPG（Deep Deterministic Policy Gradient）结构，在后面会进一步描述。

4. 状态空间定义

对于每个layer，考虑优化如下的状态空间

$$s_t = \left(t,n,c,h,w,stride,k,FLOPs[t],reduced,rest,a_{t-1} \right)$$

其中，t为layer的编号，$FLOPs[t]$为layer的FLOPs值，a_{t-1}为上一层的压缩率（此处定义为压缩至百分之多少，也即稀疏率）

而比较有趣的是$reduced$和$rest$，前者是指本层前的所有layer剪枝减掉的所有FLOPs；后者是指本层后的所有layer原始的所有FLOPs。这两个参数的存在是方便agent区分不同的layer，以便给出不同的剪枝策略。

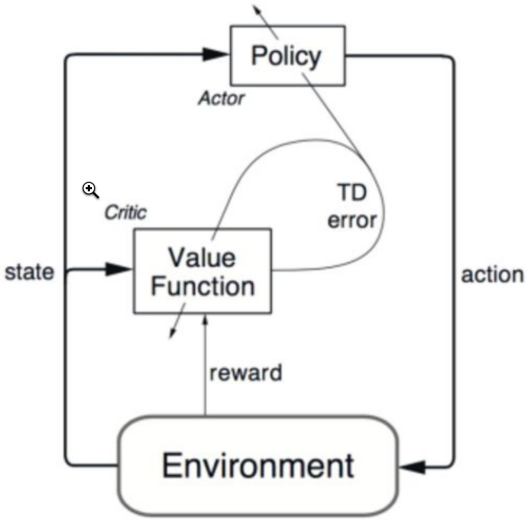
5. 强化学习agent介绍与选择

传统的NAS中，一般Action space是高度离散的，例如${64,128,256,512}$就足够覆盖一个优秀结构所需要用到的channels数量。额外增加一个65对搜索结构的优化没有太大帮助。

然而在网络压缩问题当中，网络结构对压缩率的敏感程度很高。可能0.19的压缩率网络就灾难性崩坏了，但是0.2的压缩率网络acc会基本不降。为了适应不用backbone的压缩需求，显然AMC当中的agent需要拥有连续的action space，输出一个具体的压缩率而不是从压缩率列表中选一个。这就指向了Policy Gradients结构而不是最早的DQN系列结构。

至于为什么选择Actor-Critic，莫烦有过比较精彩的论述

> 我们有了像 Q-learning这么伟大的算法, 为什么还要瞎折腾出一个 Actor-Critic? 原来 Actor-Critic 的 Actor 的前生是 Policy Gradients, 这能让它毫不费力地在连续动作中选取合适的动作, 而 Q-learning 做这件事会瘫痪. 那为什么不直接用 Policy Gradients 呢? 原来 Actor Critic 中的 Critic 的前生是 Q-learning 或者其他的 以值为基础的学习法 , 能进行单步更新, 而传统的 Policy Gradients 则是回合更新, 这降低了学习效率.

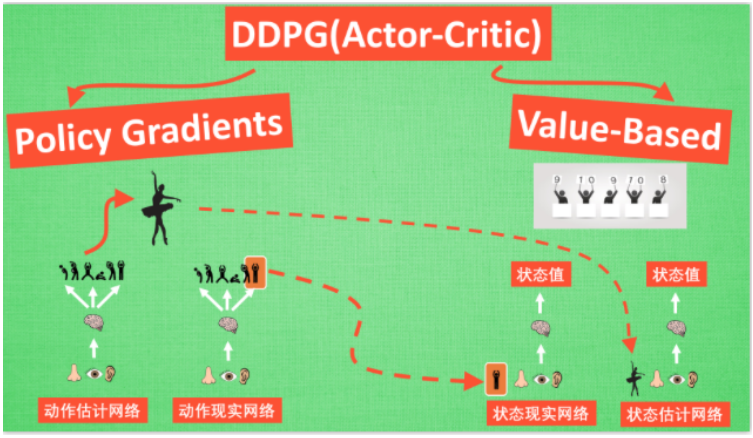


Actor-Critic算法的大概过程如下：Actor选择动作，Critic来告诉Actor它选择的动作是否合适。在这一过程中，Actor不断迭代，得到每一个状态下选择每一动作的合理概率，Critic也不断迭代，不断完善每个状态下选择每一个动作的奖惩值。

本文实际选用的DDPG和Actor-Critic原理上是一码事，DDPG是为了修正Actor-Critic存在如下的问题衍生出来的工作。

> Actor-Critic 涉及到了两个神经网络, 而且每次都是在连续状态中更新参数, 每次参数更新前后都存在相关性, 导致神经网络只能片面的看待问题, 甚至导致神经网络学不到东西

因此DDPG选择引入在电动游戏Atari上大获成功的DQN系列工作经验，采用经验池和双网络结构来优化原本的Actor-Critic架构。关于DDPG的详细原理和思路，莫烦也有精彩的讲解。



其中：

- 经验池（experience replay）是为了解决DL目标分布固定；RL的分布一直变化的问题， 能够打破参数更新的相关性。
- 而双网络结构则是维持了整体网络预测过程的稳定性，解决了非线性网络表示值函数时出现不稳定等问题。具体来说，我们只需要训练动作估计网络和状态估计网络的参数，而动作现实网络和状态现实网络的参数是由前面两个网络每隔一定的时间复制过去的

6. 两类目标的search protocal设计

Resource-Constrained Compression

实际上，本文当中没有针对资源限制修改网络中的loss，使用的是完全通用的loss函数

$$R_{err} = -Error$$

对于不同的资源限制，能够得到不同的临界压缩率。本文实现资源限制网络搜索的方式是限制action space。也即当发现后续的所有layer即使都进行毁灭性剪枝都无法满足总体的资源限制的时候，开始对action进行控制。

这样的方法有一个显著的好处，就是不同种类的资源限制都能够套用这个框架，例如FLOPs、inference时间、储存空间等等。

Accuracy-Guaranteed Compression

对于在保证acc不降的情况下尽可能压缩网络的需求，本文通过修改loss函数的方式实现。

$$R_{FLOPs} = -Error \cdot log(FLOPs)$$

$$R_{Param} = -Error \cdot log(#Param)$$

显然，这样的loss对Error是非常敏感的，训练的结果会优先保障acc性能。而FLOPs或者model size也会参与到reward的惩罚中，因此搜索的结果会倾向于高acc条件下尽可能进行模型压缩。

7. 算法伪代码

Algorithm 1 Predict the sparsity ratio $\texttt{action}_t$ for layer L_t with constrained model size (number of parameters) using fine-grained pruning

▷ Initialize the reduced model size so far
if t is equal to 0 **then**
  $\texttt{W}_{\text{reduced}} \leftarrow 0$
end if

▷ Compute the agent's action and bound it with the maximum sparsity ratio
$\texttt{action}_t \leftarrow \mu'(s_t)$
$\texttt{action}_t \leftarrow \min(\texttt{action}_t, \texttt{action}_{\max})$

▷ Compute the model size of the whole model and all the later layers
$\texttt{W}_{\text{all}} \leftarrow \sum_k \texttt{W}_k$
$\texttt{W}_{\text{rest}} \leftarrow \sum_{k=t+1} \texttt{W}_k$

▷ Compute the number of parameters we have to reduce in the current layer if all the later layers are pruned with the maximum sparsity ratio. α is the target sparsity ratio of the whole model.
$\texttt{W}_{\text{duty}} \leftarrow \alpha \cdot \texttt{W}_{\text{all}} - \texttt{action}_{\max} \cdot \texttt{W}_{\text{rest}} - \texttt{W}_{\text{reduced}}$

▷ Bound $\texttt{action}_t$ if it is too small to meet the target model size reduction
$\texttt{action}_t \leftarrow \max(\texttt{action}_t, \texttt{W}_{\text{duty}}/\texttt{W}_t)$

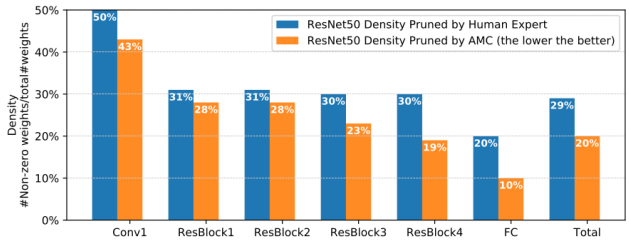
▷ Update the accumulation of reduced model size
$\texttt{W}_{\text{reduced}} \leftarrow \texttt{W}_{\text{reduced}} + \texttt{action}_t \cdot \texttt{W}_t$

return $\texttt{action}_t$

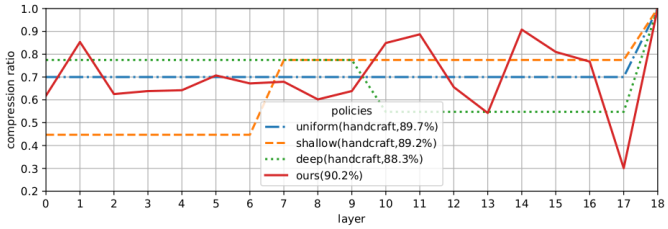
Results

Model	Policy	Ratio	Val Acc.	Test Acc.	Acc. after FT.
Plain-20 (90.5%)	deep (handcraft)	50% FLOPs	79.6	79.2	88.3
	shallow (handcraft)		83.2	82.9	89.2
	uniform (handcraft)		84.0	83.9	89.7
	AMC (R_{Err})		**86.4**	**86.0**	**90.2**
ResNet-56 (92.8%)	uniform (handcraft)	50% FLOPs	87.5	87.4	89.8
	deep (handcraft)		88.4	88.4	91.5
	AMC (R_{Err})		**90.2**	**90.1**	**91.9**
ResNet-50 (93.53%)	**AMC (R_{Param})**	60% Params	**93.64**	**93.55**	-

AMC和之前的hand-craft模型压缩方法性能对比



在ResNet50上精度不降条件下，hand-craft（压缩3.4倍）和AMC（压缩5倍）对比



利用RL算法搜索出来的各个layer压缩率鉴赏

	policy	FLOPs	ΔAcc %
VGG-16	FP (handcraft) [31]	20%	-14.6
	RNP (handcraft) [33]		-3.58
	SPP (handcraft) [49]		-2.3
	CP (handcraft) [22]		-1.7
	AMC (ours)		**-1.4**
MobileNet	uniform (0.75-224) [23]	56%	-2.5
	AMC (ours)	50%	**-0.4**
	uniform (0.75-192) [23]	41%	-3.7
	AMC (ours)	40%	**-1.7**
MobileNet-V2	uniform (0.75-224) [44]	50%	-2.0
	AMC (ours)		**-1.0**

在ImageNet上压缩VGG和MobileNet也获得成功

	mAP (%)	mAP [0.5, 0.95] (%)
baseline	68.7	36.7
2× handcrafted [22]	68.3 (-0.4)	36.7 (-0.0)
4× handcrafted [22]	66.9 (-1.8)	35.1 (-1.6)
4× handcrafted [53]	67.8 (-0.9)	36.5 (-0.2)
4× AMC (ours)	**68.8 (+0.1)**	**37.2 (+0.5)**

对检测领域的小测试：在PascalVOC2007上压缩VGG16，性能基本得到保证

Insights

最近NAS的大潮汹涌，其中DeepMind一马当先，前序Keys-5中降到的agent基本都来源于DeepMind。本文可以算是RL、NAS在模型压缩子领域的自然延伸。

在创造性方面，其实没有太多值得称道的地方。不过状态空间的设置和Resource Restricted设置的方式还是值得参考的。

最近出于不明原因，出现了很多搞这个的工作。

谷歌的自动模型压缩框架Tensorflow-Lite：

- 介绍推送
- 官方介绍
- 官方教程

腾讯的自动模型压缩框架PocketFlow：

- 介绍推送
- 代码：仍未但是据说即将开源