Neural Packet Classification

Eric Liang¹, $HangZhu^2$, $XinJin^2$, $IonStoica^1$ 1UCBerkeley , 2JohnsHopkinsUniversity

ekl@berkeley.edu, hzhu@jhu.edu, xinjin@cs.jhu.edu, istoica@cs.berkeley.edu

摘要

数据包分类是计算机网络中的一个基本问题。这个问题暴露了计算和状态复杂性之间的艰难权衡,这使得它特别具有挑战性。为了解决这个问题,现有的解决方案依赖于复杂的手工调整的启发式方法,这些方法难以优化。本文提出了一种深度强化学习(RL)的方法来解决数据包分类问题。NeuroCuts可以为一组给定的规则和目标学习优化数据包分类,它产生了针对特定规则集和给定性能指标(如分类时间、内存占用或两者的组合)进行优化的紧凑决策树。

关键词:数据包分类;强化学习;深度学习;决策树

1 引言

数据包分类是计算机网络中的一个基本但也具有挑战性的问题。数据包分类的目标是将一个给定的数据包与一组规则相匹配,并在优化分类时间和/或内存占用的同时做到这一点。

现有的解决方案依赖手工调整的启发式方法,通常是构建决策树。缺点一是手工调整的放肆使得难以优化不同的规则集,因为需要昂贵的成本;缺点二是这些启发式方法并不明确针对一个给定目标优化,可能导致性能不佳。

本次课程的论文复现工作重点是利用深度强化学习(Deep Reinforcement Learning, DRL)来构建决策树。RL 具有稀疏和延迟奖励的特性,并且可以明确地优化性能目标,因此很适合数据包分类问题。它可以提高数据包分类的准确性,且目标是最小化分类时间和内存占用。

2 相关工作

数据包分类是许多网络功能的关键组成部分,包括防火墙^[1]、访问控制、流量工程和网络测量。 因此,数据包分类器被企业、云供应商、ISP 和 IXP 广泛部署^{[2][3]}。一个数据包分类器包含一个规则列 表,如图 1所示每个规则都指定了数据包头中多个字段的模式,这些字段包括源和目的 IP 地址,源和 目的端口号,以及协议类型。规则的模式指定哪些数据包符合该规则。以下介绍现有的两种数据包分 类解决方案。

Priority	Src IP	Dst IP	Src Port	Dst Port	Protocol
2	10.0.0.0	10.0.0.0/16	*	*	*
1	*	*	[0, 1023]	[0, 1023]	TCP
0	*	*	*	*	*

图 1: 一个数据包分类器的例子

2.1 基于硬件的数据包分类方案

基于硬件的数据包分类方案利用三元内容可寻址存储器(Ternary Content Addressable Memory, TCAMs)将所有规则存储在一个关联存储器中,然后将一个数据包与所有这些规则并行匹配^[4]。虽然 TCAMs 提供了恒定的分类时间,但由于 TCAM 本身很复杂,这种复杂性导致了较高的成本和功耗,所以基于 TCAM 的解决方案对于实现大型分类器来说并不可行^[5]。

2.2 基于软件的数据包分类方案

基于软件的数据包分类方案通常是构建决策树^[6],虽然这种解决方案比基于 TCAM 的解决方案 更具可扩展性,但它由于分类操作需要从根到匹配的叶子遍历决策树,所以速度会比较慢。如何构建 一棵高效的决策树是需要关注的重点,但目前的研究都是依赖手工调整的启发式方法构建决策树。如 果一个启发式方法过于笼统,它就不能利用特定规则集的特点,如果一个启发式方法是为某个特定规 则集设计的,那它的泛化性就很差,无法理解和优化不同的规则集。

3 本文方法

3.1 本文方法概述

我们的解决方案使用 DL 来构建高效的决策树。有三个特点使 RL 适用于数据包分类。首先,建立决策树的自然解决方案是从一个节点开始,递归地切割节点。但这种方法并没有一个贪婪的解决方案。当做出切割一个节点的决定时,在树构建完之前,我们并不会知道这个决定是否是一个好的决定(即它是否导致了一个有效的树)。RL 捕捉到了这个特点,因为在强化学习中,一个动作对性能目标的影响不是能立即知道的。第二,现有的启发式算法所采取的行动与性能目标只有松散的关系,RL 算法则是以直接最大化性能为目标。第三,数据包分类问题中所构建的 RL 模型是可以被快速评估的。通过快速评估每个模型,我们大大减少了学习时间。基于以上特点,本文设计了 NeuroCuts。

3.2 NeuroCuts 框架

NeuroCuts 作为一个 RL 系统的框架如图 2所示。环境由规则集和当前的决策树组成,而代理使用一个模型(由一个 DNN 实现),旨在选择最佳的切割或分割动作来逐步建立树。切割动作将一个节点沿着选定的维度(即 SrcIP、DstIP、SrcPort、DstPort 和 Protocol 中的一个)分成若干个子范围(即 2、4、8、16 或 32 个范围),并在树上创建那么多的子节点。另一方面,分区动作将一个节点的规则分为不相干的子集(例如,基于一个维度的覆盖率),并为每个子集创建一个新的子节点。当前节点的可用行动在每一步都由环境公布,代理在这些行动中进行选择以生成树,随着时间的推移,代理人学会了优化其决策,以此最大化来自环境的奖励。

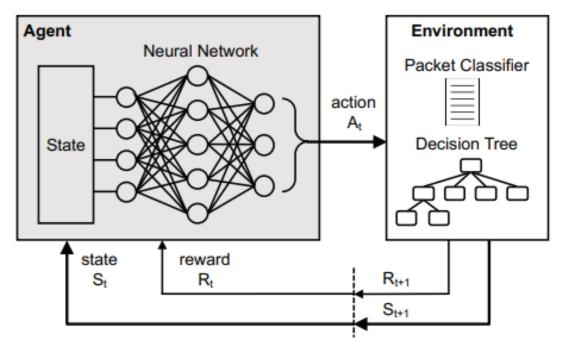


图 2: NeuroCuts 框架图

3.3 NeuroCuts 训练算法

RL 算法的目标是计算一个策略,使环境中的奖励最大化。代理从一个初始策略开始,用多次滚动来评估它,然后根据这些滚动的结果(奖励)来更新它。然后,它重复这个过程,直到对奖励满意为止。

本文使用 actor-crtic 算法^[7]来训练代理的策略。NeuroCuts 首先初始化所有的参数,然后运行 N 次滚动来训练策略和价值函数。其中,在每次滚动开始时,它将决策树重新初始化为根节点。然后它根据当前的政策,通过反复选择和应用每个非终端叶子节点的行动来增量地建立树。终端叶子节点是一个规则数量低于给定阈值的节点。决策树建立后,重置梯度,然后算法在所有树节点上迭代,以汇总梯度。最后,NeuroCuts 使用梯度来更新 actor-crtic 网络的参数,并继续进行下一个滚动。

4 复现细节

4.1 与已有开源代码对比

该文献的作者在 GitHub 上提供了开源代码。本次复现引用了源代码中的数据结构定义模块以及 RL 模型训练模块。数据结构定义模块包括定义了规则、节点、决策树的结构以及 DNN 模型,如图 3、4所示。RL 模型训练模块重点包括 RL 环境的定义,如图 5所示。



图 3: 规则类 (a); 节点类 (b)

图 4: 决策树类 (a); DNN 模型 (b)

```
lass NeuroCutsEnv(MultiAgentEnv):
     "NeuroCuts multi-agent tree building environment.
  different agent. All the agents share the same policy. We
  aggregate rewards at the end of the episode and assign each cut its reward based on the policy performance (actual depth).
在此环境中,树中的每个"初割"都是由不同的代理执行的操作。所有代理共享相同的策略。
   我们在每集结束时汇总奖励,并根据策略性能(实际深度)分配每个"切割"的奖励。
  NeuroCutsEnv継承MultiAgentEnv
  基本框架要按照MultiAgentEnv设置:
  class NeuroCutsEnv(MultiAgentEnv):
      def __init__(self,env_config):
    self.action_space<<Space>
    self.observation_space<<Space>
       def reset(self):
      def step(self,action):
           return <obs>,<reward:float>,<done:bool>,<info:dict>
                rules_file,
                leaf_threshold=16,
                max_cuts_per_dimension=5,
max_actions_per_episode=5000,
                 max_depth=100,
                partition_mode=None, #由用户在run_neurocuts.py的时候给出
                 reward_shape="linear",
                depth_weight=1.0,
                dump_dir=None,
                 #GAE(eneralized advantage estimation)泛化优势估计,将λ-return用于估计优势函数,可以平衡优势函数估计中的偏差和方差
                 #一部分策略梯度(Policy Gradient)经常会选择优势函数来构造策略梯度
                 #GAE由参数γ∈[0,1]λ∈[0,1]来表示
                tree_gae=True,
                 tree_gae_gamma=1.0,
```

图 5: RL 环境

4.2 实验环境搭建

本次实验的环境是 Ubuntu20.04, TensorFlow 使用版本 1.14.0。由于对于非常大的规则集,训练时间会很长,所以利用分布式 RL 库 Rllib 来解决这个问题。ray 使用版本 0.7.6。本次实验使用的数据集是 Classbench 规则集,其中包括了若干 ACL、FW 和 IPC 三个不同规模的规则集。

4.3 界面分析与使用说明

首先,输入如图 6所示的命令来训练一棵决策树:其中-rules 属性表示选用哪个或哪几个规则集来构建决策树,-fast 表示使用快速超参数配置,-dump-dir 属性表示将有效的树转储到指定目录以供以后检查。其余可选属性还有:-num-workers 表示要从 RLlib 请求的并行工作线程数,-partition-mode表示设置分区器等。

```
cdt@cdt-HP-Pavilion-Laptop-14-ce0xxx; ~/neurocuts-docker-master
neurocuts_env.py
naster$ python ru
naster$ ^C(pv37)
                                                    run_neurocuts.py test.py Visualize.ipynb
--rules=acl4_1k --fast --dump-dir=/home/cdt/tree
         python run_neurocuts.py
```

图 6: 运行命令训练决策树

接下来,输入如图 7所示命令在上述的指定目录中查看训练过程中生成的树,树的信息包括树的 节点个数、平均每个规则占用的内存以及最大访存次数等。

```
ree has 990 rules
itats: { bytes_per_rule': 453.2121212121212, 'memory_access': 32, 'num_leaf_node': 4061, 'num_nonleaf_node': 717, 'num_node': 4778}
ible visualization:
```

图 7: 查看生成的树的信息

最后,输入如图 8所示命令在 tensorboard 中查看训练过程中保存的数据。

```
1,他人知官 O川内印文在 ICESOFOORIU 中国自 则练及工作体力的数据。

which is a more content of the provided as (type, (1,)) / '(1,) type'.

producted and condad fenvs / py37/lib/python3.7/site-packages/tensorboard/compat/tensorflow_stub/dtypes.py:542: FutureWarning: Passing (type on of numpy, it will be understood as (type, (1,)) / '(1,) type'.

producted and an environment of the product of the produc
    on of numpy, it will be understood as (type, (1,)) / '(1,)type'.

np_qint32 = np.dtype([("qint32", np.int32, 1)])

nme/cdt/anaconda3/envs/py37/lib/python3.7/site-packages/tensorboard/compat/tensorflow_stub/dtypes.py:550: FutureWarning: Passing (typon of numpy, it will be understood as (type, (1,)) / '(1,)type'.

np_resource = np.dtype([("resource", np.ubyte, 1)])
```

图 8: 查看走势图

实验结果分析 5

本次实验针对 acl4 1k 规则集构建决策树,该规则集中一共有 990 个规则。在指定目录下我们可 以看到,在训练过程中一共生成了5棵树,如图9所示。

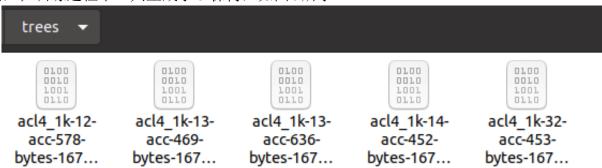


图 9: 生成决策树的过程中一共生成了 5 棵有效的树

由于 NeuroCuts 是从一个随机的初始化策略开始,所以一开始的树形状并不好。如图 10所示,生 成的第一棵树的深度为 32。随着时间的推移, NeuroCuts 学会减少树的深度并制定更好的策略。最后 生成的决策树的深度为12,如图11所示。

图 10: 在 acl4 1k 规则集上训练出的第一棵决策树

```
cdt@cdt-HP-Pavilion-Laptop-14-ce0xxx:-/neurocuts-docker-master

(py37) cdte2cdt-IP-Pavilion-Laptop-14-ce0xxx:-/neurocuts-docker-master  
python inspect_tree.py /home/cdt/trees/acl4_1k-12-acc-578-bytes-1671780932.4067135.pkl
This tree has 900 rules
Tree stats: ('bytes_per_rule': 578.6222222222223, 'memory_access': 12, 'num_leaf_node': 5325, 'num_nonleaf_node': 847, 'num_node': 6172}
Plottable visualization:
widths, 1,3,19,59,293,742,1314,1643,1153,606,312,27
dim0,0,0,0,6,48,104,120,78,38,21,0,0
dim1,0,1,4,1,11,35,49,58,75,62,40,0
dim1,0,1,4,1,13,54,95,87,56,24,0,0
dim2,0,0,0,0,0,0,0,0,0,0,0,0,0,0
dim3,0,0,4,15,20,13,18,4,1,0,0,0
dim3,0,0,4,15,20,13,18,4,1,0,0,0
dim4,1,2,0,0,0,0,0,0,0,0
```

图 11: 在 acl4 1k 规则集上训练出的最终决策树

通过 tensorboard 中的走势图,如图 12所示,我们可以看出各种数据在训练过程中的变化。



图 12: 走势图

6 总结与展望

本文实现了基于学习的数据包分类算法 NeuroCuts。NeuroCuts 可以对最坏情况下的分类时间或内存占用进行了优化,并能简单高效地针对特定规则集生成决策树。在未来的研究中,通过考虑特定的流量模式,NeuroCuts 可以扩展到其他目标,如平均分类时间。这将使 NeuroCuts 不仅能对特定的分类器进行优化,而且能对特定部署中的特定流量模式进行优化。

参考文献

- [1] GUPTA P. Packet Classification using Hierarchical Intelligent Cuttings[J]. Proc.hot Interconnects VII Aug, 1999: 34-41.
- [2] BABOESCU F, SINGH S, VARGHESE G. Packet classification for core routers: is there an alternative to CAMs?[C]//IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No.03CH37428): vol. 1. 2003: 53-63 vol.1. DOI: 10.1109/IN

- FCOM.2003.1208658.
- [3] SPITZNAGEL E, TAYLOR D, TURNER J. Packet classification using extended TCAMs[C]//11th IEEE International Conference on Network Protocols, 2003. Proceedings. 2003: 120-131. DOI: 10.1109/ICNP .2003.1249762.
- [4] LAKSHMINARAYANAN K, RANGARAJAN A, VENKATACHARY S. Algorithms for advanced packet classification with ternary CAMs[C]//Proceedings of the ACM SIGCOMM 2005 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Philadelphia, Pennsylvania, USA, August 22-26, 2005. 2005.
- [5] VAMANAN B, VOSKUILEN G, VIJAYKUMAR T N. EffiCuts: Optimizing Packet Classification for Memory and Throughput[J/OL]. SIGCOMM Comput. Commun. Rev., 2010, 40(4): 207-218. https://doi.org/10.1145/1851275.1851208. DOI: 10.1145/1851275.1851208.
- [6] LI W, LI X, LI H, et al. CutSplit: A Decision-Tree Combining Cutting and Splitting for Scalable Packet Classification[C]//IEEE INFOCOM 2018 IEEE Conference on Computer Communications. 2018: 2645-2653. DOI: 10.1109/INFOCOM.2018.8485947.
- [7] KONDA V R, TSITSIKLIS N. Actor-Critic Algorithms[J]., 2001.