Summary of "Neural Packet Classification"

Jingyuan Zhu (jingyz), Yuze Lou (yuzelou)

Problem and Motivation

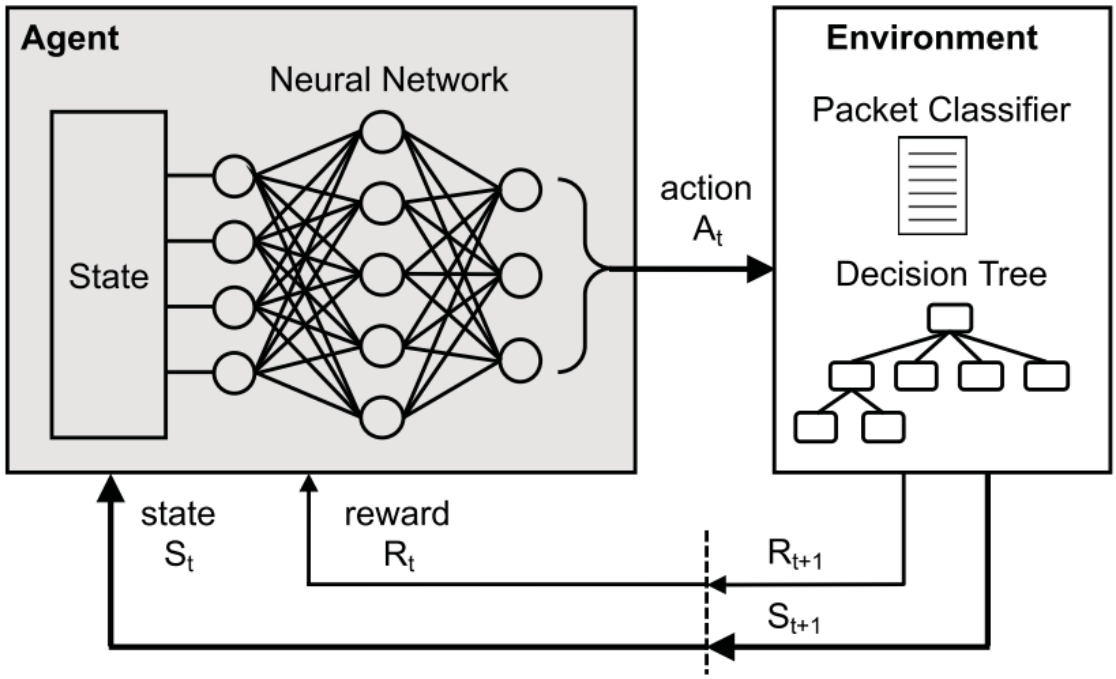
Packet classification is a process aims to match a rule in the set of rules for every coming packets. It is a fundamental problem in computer networking, where people have to deal with the tradeoff between computation and state complexity. Existing solutions in software usually rely on hand - tuned heuristics, which is ad-hoc and loosely related to the objective. Recently, deep reinforcement learning has been emerging, and it is aiming at maximize the objective by interacting with the environments. This makes RL a good fit for the problem of neural classification at the solution of building a decision tree. Because build decision tree can not be done with the greedy approach, and RL also assumes no impact known for a single given steps. In addition, evaluating the performance of neural classification is very quick, thus it avoids the problem of evaluation in traditional RL.

Hypothesis

NeuroCuts is a system that uses RL to generate a decision tree to maximize the specified objective. The reason for it to adopt RL instead of a traditional neural network in order to have an end-to-end solution is that NeuroCuts is based on the hypothesis of neural classification cannot make any mistakes. Making mistakes could be acceptable in some applications but not all the time.

Solution Overview

NeuroCuts aims at using RL to generate decision tree that represents the classification rule and maximize the objectives (classification time, memory). The figure illustrate the architecture of NeuroCuts.



The environment is basically the packet classification rules and decision tree to generate. At every state, the agent will decide on the action, which is which node to split next on which properties (src/dest IP, Port, Protocol). At the end of 1 iteration, the environment will provide the reward (objective) for the agent to optimize on its policy.

For the state representation, it takes advantage of the underlying structure of packet classification trees to design the state as only the current node, since the decision (action) to do next does not depend on the previous node. Therefore, the size of the state is largely saved and the state can now be encoded as a fixed length input.

In terms of RL learning algorithms, NeuroCuts adopt a MDP as the strawman approach, and use an actor-critic algorithm to train the policy. By experiments it turns out that it is more effective than the Q-Learning.

In order to speed up the training, NeuroCuts also proposes some optimizations. It uses rollout truncation to prune out some of the steps to prevent large initial trees. It also adopts PPO (actor-critic algorithm with entropy regularization and a clipped surrogate objective) to speed up the exploration.

Limitations and Possible Improvements

- Currently, NeuroCuts only assumes the classification rules are from IP, Port and protocol. Potentially it can support more (unfixed) attributes. For example, the packet classification includes Deep Packet Inspection, which inspect further into the packet headers.
- It remains unclear that whether the environment NeuroCuts is trying to learning similar enough to the real world's case, in terms of memory and CPU. It can develop some mechanism for online learning where the real environment is involved after some offline training.

Summary of Class Discussion

- Will the generation of decision tree causing overfit?
    - Yes, same rules could be replicated in multiple nodes.
- Are there anything that could be improved from the system perspective?

    - Once the classification rules are changed, the decision tree has to be re-trained. How to efficient re-trained or incrementally trained could be an improvements
- Can this decision tree be encoded into some other models like linear model?
    - Yes, as long as you can encode the decision tree.

Summary of "Learning Scheduling Algorithms for Data Processing Clusters"

Yuze Lou (yuzelou), Jingyuan Zhu (jingyz)

Problem and Motivation

Efficiently scheduling data processing jobs on distributed clusters requires complex algorithms. Current systems use relatively simple and generalized heuristics, however, ignoring optimization opportunities that require complex workload-specific algorithms. For enterprises, even small improvements in utilization can save millions of dollars at scale.

Manually developing algorithms for every workload is not only error-prone but also could be expensive. An ideal solution has to be an automated one.

Hypothesis

1. A workload of the jobs is available for training.
2. The implementation and experiments in the paper are in the context of Spark.

Solution Overview

Given a workload of jobs, the system generates three levels of embedding: per-node embeddings, per-job embeddings, and a global embeddings. Based on these embeddings and a cluster simulator, an RL model is trained to learn the optimal policy for this workload.

Then in development, jobs are scheduled according to the learned policy.

In this paper, each job is in DAG format, where each node represents a stage, and an arrow represents data flow and dependency.

Per-Node Embeddings

Per-node embeddings capture information about the node and its children.

To compute these vectors, Decima propagates information from children to parent nodes in a sequence of message passing steps, starting from the leaves of the DAG by aggregating all the embeddings of descendants using a non-linear function and its stage attributes feature vector.

Per-Job and Global Embeddings

The graph neural network also computes a summary of all node embeddings for each DAG, and a global summary across all DAGs, by adding links from all nodes in a job to that node's summary node, and adding links from all job summary nodes to the global summary node.

Then the embeddings are calculated in the same way as the per-node embeddings.

Encoding scheduling decisions as actions

Scheduling decisions in Decima are decomposed into a series of two-dimensional actions which output (i) a stage designated to be scheduled next, and (ii) an upper limit on the number of executors to use for that stage's job.

Stage Selection

For each node in a job, the policy networks computes a score that maps the embedding vectors to a scalar value.

Then Decima uses a SoftMax operation to compute the probability of selecting node v based on the priority scores.

Parallelism Limit Selection

For each job i, Decima's policy network also computes a score for assigning parallelism limit l to job i, using another score function. Similar to stage selection, Decima applies a soft- max operation on these scores to compute the probability of selecting a parallelism limit.

Importantly,Decima uses the same score function w(·) for all jobs and all parallelism limit value.

Limitations and Possible Improvements

1. A workload of jobs have to be available for training. And later in the development, the incoming workload has to be similar the one used in training.
2. Data locality is not considered.

Summary of Class Discussion

1. Data locality is considered as a future direction.
2. Someone asked about whether Decima requires task durations as an input feature to the Neural Network. According to the paper, Decima indeed requires avg. task durations and the number of tasks in a stage.