Paper: Neural Combinatorial Optimization with Reinforcement Learning

Summary

In this paper, the authors present a framework to tackle combinatorial optimization problems using reinforcement learning and neural network. They specifically focus on traveling salesman problem (TSP) but claim that their solution can be applied to other NP-hard problems such as Knapsack with minimal adjustments. In general, TSP solvers rely on handcrafted heuristics that guide search procedures to find competitive tours effectively. Such procedures tend to work well, however, it is not very generalizable, even when the same problem is changed slightly. In contrast, machine learning methods tend to be useful as they automatically discover their ow heuristics based on the training data. In most cases, supervised learning does not apply to most combinatorial optimization because it is very hard to access optimal labels. However, one can compare the quality of a set of solutions using a verifier and provide some reward feedbacks to a learning algorithm. Therefore, the reinforcement learning paradigm seems to be the best, which is what the author uses.

The authors focus on 2D Euclidean TSP in this paper. Given an input graph, represented as a sequence of n cities in a two-dimensional space, the authors are concerned with finishing a permutation of the points, called the tour, that visits each city once and has the minimum total length. They aim to learn the parameters of a stochastic policy that given an input set of points s, assigns high probability to short tours and low probabilities to long tours. They address the two major issues with using vanilla sequence to sequence model to address the TSP where the output vocabulary is {1, 2, 3, …, n}: inability to generalize to inputs with more than n cities; requires access to ground-truth output permutations to optimize the parameters with conditional log-likelihood.

For generalization beyond a pre-specified graph-size, they use a pointer network, which makes use of a set of non-parametric softmax modules, resembling the attention mechanism. This allows the model to effectively point to a specific position in the input sequence rather than predicting an index value from a fixed-size vocabulary. Their pointer network comprises two RNN modules, encoder and decoder, both of which consist of LSTM cells. They optimize the parameters of a pointer network using model-free policy-based Reinforcement Learning. They use policy gradient methods and stochastic gradient descent specifically. The gradient is formulated using the REINFORCE algorithm. For learning the expected tour length, the use of an auxiliary network called a critic.

The authors consider two search strategies: sampling and active search. Under-sampling, they simply sample multiple candidate tours from their stochastic policy and select the shortest one. Under active searching, they refine parameters of the stochastic policy during inference to minimize on a single test input. They train two models, RL pretraining-Active Search, and Active Search. The first one starts from a trained model whereas the second one from an untrained model.

They conducted experiments on Euclidean TSP20, 50, and 100. They compared their model against 3 different baselines: Christofides, the vehicle routing solver from OR-Tools, and optimal. They also trained four of their models: RL pretraining-Greedy, Active Search, RL pretraining-Sampling, and RL pretraining-Active Search. Notably, results demonstrate that training with RL significantly improves over supervised learning. All their models comfortably

surpass Christofides' heuristic. Searching at inference time proves crucial to get closer to optimality but comes at the expense of longer running times. RL pretraining-Sampling and RL pretraining-Active Search are the most competitive Neural Combinatorial Optimization methods and recover the optimal solution in a significant number of their test cases.

The authors also applied their Neural Combinatorial Optimization to another problem called the 0-1 Knapsack problem. They present the performances of RL pretraining-Greedy and Active Search and compare them with two simple baselines: heuristic-based greedy approach and random search. They found out that RL pretraining-Greedy yields solutions that, on average, are just 1% less than optimal and Active Search solves all instances to optimality.

Strengths

- The paper is very well organized with nicely explained sections and subsections.
- They have also included the previous work section which I felt is particularly helpful in understanding the field.
- I also liked the fact that they explained how their model works against another optimization problem such as Knapsack.

Weaknesses

- I felt the conclusion was too short. They could have included a discussion section and wrap things up nicely.

Confusions

- I did not fully understand their actor-critic training algorithm.
- I was confused about the REINFORCE algorithm that they use to formulate gradient.

Discussions

- Can we discuss more policy gradients?
- How can one be able to apply this model to other combinatorial problems? What necessary adjustments will be required?
- Can other reinforcement learning algorithms be used for combinatorial optimization problems?