Neural Combinatorial Optimization With Reinforcement Learning

CS885 Reinforcement Learning

Paper by Bello, I., Pham, H., Le, Q. V., Norouzi, M., & Bengio, S. (2016)

Presented by Yan Shi

UNIVERSITY OF WATERLOO

Outline

1. Introduction
2. Background
3. Algorithms and optimization
4. Experiments
5. Conclusions

Introduction

Travelling Salesman Problem

- Combinatorial Optimization is a fundamental problem in computer science
- Travelling Salesman Problem is such a typical problem and is NP hard, where given a graph, one needs to search the space of permutations to find an optimal sequence of nodes with minimal total edge weights (tour length).
- In 2D Euclidean space, nodes are 2D points and edge weights are Euclidean distances between pairs of points.

Introduction

Target & Solution

- This paper will use reinforcement learning and neural networks to tackle the combinatorial optimization problem, especially TSP.
- We want to train a recurrent neural network such that, given a set of city coordinates, it will predict a distribution over different cities permutations.
- The recurrent neural network encodes a policy and is optimized by policy gradient, where the reward signal is the negative tour length.
- We propose two main approaches, *RL Pretraining and Active Search*

Background

- The Traveling Salesman Problem is a well studied combinatorial optimization problem and many exact or approximate algorithms have been proposed.
- Like Christofides, Concorde, Google's vehicle routing problem solver
- The real challenge is applying existing search heuristics to newly encountered problems, researcher used "hyper-heuristics" to generalize their optimization system, but more or less, human created heuristic is needed.

Background

- The earliest solution for TSP using machine learning is Hopfield networks (Hopfield & Tank, 1985), but it is sensitive to hyperparameters and parameter initialization.
- Later research include applying Elastic Net (Durbin, 1987), Self Organizing Map (Fort, 1988) to TSP
- Most of the other works were analyzing and modifying the above methods, and they showed that neural network were beat by algorithmic solutions

Background

- Due to sequence to sequence learning, neural network is again the subject of study for optimization in various domain.
- In particular, the TSP is revisited in the introduction of Pointer network (Vinyals et al, 2015b), where recurrent neural network is trained in a supervised way to predict the sequence of visited cities.

UNIVERSITY OF WATERLOO

Algorithm and Optimization

Construction

- We focus on a 2D Euclidean TSP. And let the input be the sequence of cities (points) $s = \{x_i\}_{i=1}^{n}$, where each $x_i \in \mathbb{R}^2$.
- The target is to find a permutation π of these points, terms as a tour, that visits each city and has minimum length.
- Define the length of a tour π as:

$$L(\pi|s) = \left\|x_{\pi(n)} - x_{\pi(1)}\right\|_2 + \sum_{i=1}^{n-1}\left\|x_{\pi(i+1)} - x_{\pi(i)}\right\|_2$$

Algorithm and Optimization

Construction

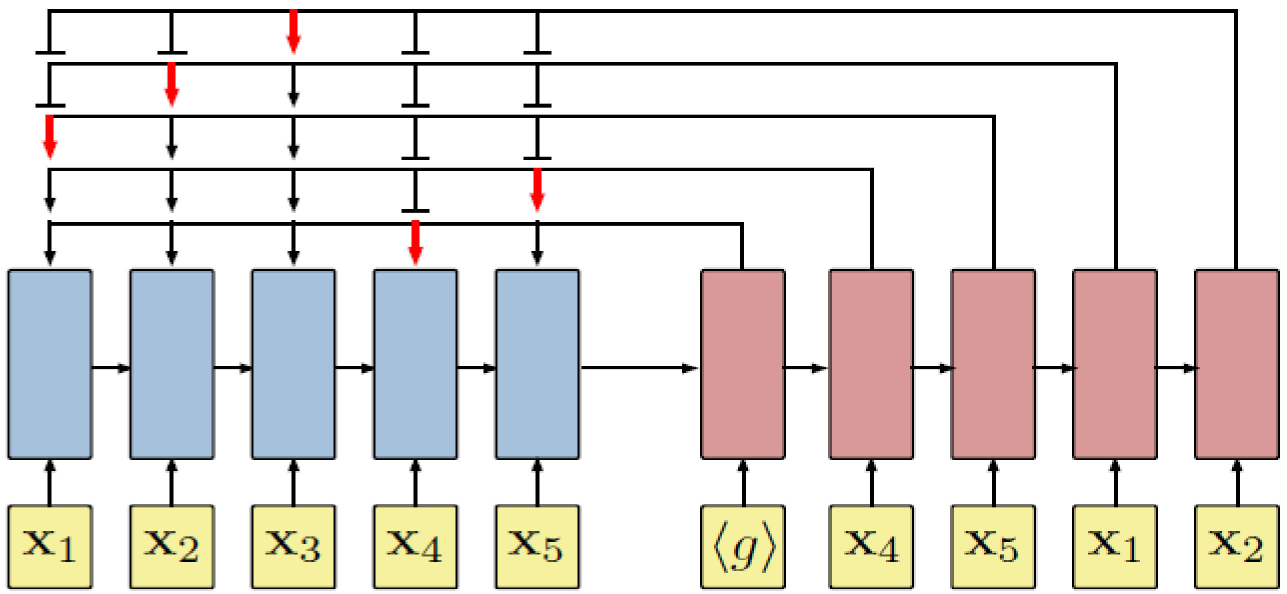
- Construct a model-free and policy based algorithm
- The goal is to learn the parameters of the stochastic policy $p(\pi|s) = \prod_{i=1}^{n} p((\pi(i)|(\pi(< i), s))$
- This stochastic policy needs to:

i. Be sequence to sequence

ii. Be generalized to different graph size

Algorithm and Optimization

Pointer network



Encoder: reads the input sequence s, one city at a time, and transforms it into a sequence of latent memory states $\{enc_i\}_{i=1}^{n}$, and each $enc_i \in \mathbb{R}^d$

Decoder: uses a pointing mechanism to produce a distribution over the next city to visit in the tour.

$$u_i = \begin{cases} v^T \tanh\left(W_{enc} enc_i + W_{dec} dec_j\right) & if\ i \neq \pi(k)\ for\ all\ k < i \\ -\infty & otherwise \end{cases}$$

$$A\left(enc, dec_j; W_{enc}, W_{dec}, v\right) \stackrel{\text{def}}{=} softmax(u)$$

UNIVERSITY OF WATERLOO

Algorithm and Optimization

Optimization

- Target (loss) function

$$J(\theta|s) = \mathbb{E}_{\pi\sim p_\theta(\cdot|s)} L(\pi|s)$$

- Policy gradient with a baseline

$$\nabla_\theta J(\theta|s) = \mathbb{E}_{\pi\sim p_\theta(\cdot|s)}[\big(L(\pi|s) - b(s)\big)\nabla_\theta \log p_\theta(\cdot\,|s)]$$

- Using samples of size B to approximate expectation

$$\nabla_\theta J(\theta|s) = \frac{1}{B}\sum_{i=1}^{B}[\big(L(\pi_i|s_i) - b(s_i)\big)\nabla_\theta \log p_\theta(\pi_i|s_i)]$$

Algorithm and Optimization

Actor Critic

- Here, Let $b(s)$ (the baseline) be the expected tour length $\mathbb{E}_{\pi\sim p_\theta(\cdot|s)}[L(\pi|s)]$
- Introduce another network, called critic and parameterized by θ_v to encode $b_{\theta_v}(s)$.
- This critic network is trained along with the policy network, and the objective is

$$\mathcal{L}(\theta_v) = \frac{1}{B}\sum_{i=i}^{B}\left\|b_{\theta_v}(s) - L(\pi_i|s_i)\right\|_2^2$$

Algorithm and Optimization

Critic's Architecture

I. One LSTM encoder, similar to the pointer network, encodes the sequence of cities s to a series of latent memory states and a hidden state h

II. One LSTM processor, which takes the hidden state h as an input, process it P times, then pass to decoder

III. A two-layer ReLU neural network decoder, transforms the above output hidden state into a baseline prediction.

Algorithm and Optimization

Algorithm 1 Actor-critic training

procedure TRAIN(training set S, number of training steps T, batch size B)
    Initialize pointer network params θ
    Initialize critic network params θ_v
    for $t = 1$ to T **do**
        $s_i \sim \text{SAMPLEINPUT}(S)$ for $i \in \{1, \ldots, B\}$
        $\pi_i \sim \text{SAMPLESOLUTION}(p_\theta(.|s_i))$ for $i \in \{1, \ldots, B\}$
        $b_i \leftarrow b_{\theta_v}(s_i)$ for $i \in \{1, \ldots, B\}$
        $g_\theta \leftarrow \frac{1}{B} \sum_{i=1}^{B} (L(\pi_i|s_i) - b_i) \nabla_\theta \log p_\theta(\pi_i|s_i)$
        $\mathcal{L}_v \leftarrow \frac{1}{B} \sum_{i=1}^{B} \|b_i - L(\pi_i)\|_2^2$
        $\theta \leftarrow \text{ADAM}(\theta, g_\theta)$
        $\theta_v \leftarrow \text{ADAM}(\theta_v, \nabla_{\theta_v} \mathcal{L}_v)$
    end for
    return θ
end procedure

Algorithm and Optimization

Search Strategy

- In Algorithm 1, we were using greedy decoding at each step to select cities, but we can also sample different tours then select the shortest one.

$$A(ref, q, T; W_{ref}, W_q, v) \stackrel{\text{def}}{=} softmax(u/T)$$

- What about developing a search strategy that is not pre-trained, and will optimize parameter for every single test input?

Algorithm and Optimization

Algorithm 2 Active Search

procedure ACTIVESEARCH(input s, θ, number of candidates K, B, α)
  $\pi \leftarrow$ RANDOMSOLUTION()
  $L_\pi \leftarrow L(\pi \mid s)$
  $n \leftarrow \lceil \frac{K}{B} \rceil$
  for $t = 1 \ldots n$ **do**
    $\pi_i \sim$ SAMPLESOLUTION($p_\theta(. \mid s)$) for $i \in \{1, \ldots, B\}$
    $j \leftarrow$ ARGMIN($L(\pi_1 \mid s) \ldots L(\pi_B \mid s)$)
    $L_j \leftarrow L(\pi_j \mid s)$
    if $L_j < L_\pi$ **then**
      $\pi \leftarrow \pi_j$
      $L_\pi \leftarrow L_j$
    end if
    $g_\theta \leftarrow \frac{1}{B} \sum_{i=1}^{B} (L(\pi_i \mid s) - b) \nabla_\theta \log p_\theta(\pi_i \mid s)$
    $\theta \leftarrow$ ADAM(θ, g_θ)
    $b \leftarrow \alpha \times b + (1 - \alpha) \times (\frac{1}{B} \sum_{i=1}^{B} b_i)$
  end for
  return π
end procedure

Sample n solutions and select the shortest one

Same policy gradient as before

No critic network, using a exp moving average baseline instead

Experiment

- We consider three benchmark tasks, Euclidean TSP20, 50 and 100, for which we generate a test set of 1000 graphs. Points are drawn uniformly at random in the unit square [0, 1]
- Four target algorithms:

i. RL pretraining (Actor Critic) with greedy decoding

ii. RL pretraining (Actor Critic) with sampling

iii. RL pretraining-Active Search (run Active Search with a pretrained RL model)

iv. Active Search

Experiment

Table 1: Different learning configurations.

Configuration	Learn on training data	Sampling on test set	Refining on test set
RL pretraining-Greedy	Yes	No	No
Active Search (AS)	No	Yes	Yes
RL pretraining-Sampling	Yes	Yes	No
RL pretraining-Active Search	Yes	Yes	Yes

Experiment

- Using 3 algorithmic solutions as baselines:

i. Christofides

ii. the vehicle routing solver from OR-Tools

iii. Optimality

- For the purpose of comparison, we also trained pointer networks with the same architecture by supervised learning method (providing with the true label).

Experiment

Averaged tour length

Table 2: Average tour lengths (lower is better). Results marked (†) are from (Vinyals et al., 2015b).

Task	Supervised Learning	RL pretraining				AS	Christo -fides	OR Tools' local search	Optimal
		greedy	greedy@16	sampling	AS				
TSP20	3.88(†)	3.89	–	3.82	3.82	3.96	4.30	3.85	3.82
TSP50	6.09(†)	5.95	5.80	5.70	5.70	5.87	6.62	5.80	5.68
TSP100	10.81	8.30	7.97	7.88	7.83	8.19	9.18	7.99	7.77

Experiment

Running time

Table 3: Running times in seconds (s) of greedy methods compared to OR Tool's local search and solvers that find the optimal solutions. Time is measured over the entire test set and averaged.

Task	RL pretraining		OR-Tools'	Optimal	
	greedy	greedy@16	local search	Concorde	LK-H
TSP50	0.003s	0.04s	0.02*s*	0.05s	0.14s
TSP100	0.01s	0.15s	0.10*s*	0.22s	0.88s

Experiment

Reinforcement Learning methods

Table 4: Average tour lengths of RL pretraining-Sampling and RL pretraining-Active Search as they sample more solutions. Corresponding running times on a single Tesla K80 GPU are in parantheses.

Task	# Solutions	RL pretraining		
		Sampling $T = 1$	Sampling $T = T^*$	Active Search
TSP50	128	5.80 (3.4s)	5.80 (3.4s)	5.80 (0.5s)
	1,280	5.77 (3.4s)	5.75 (3.4s)	5.76 (5s)
	12,800	5.75 (13.8s)	5.73 (13.8s)	5.74 (50s)
	128,000	5.73 (110s)	5.71 (110s)	5.72 (500s)
	1,280,000	5.72 (1080s)	5.70 (1080s)	5.70 (5000s)
TSP100	128	8.05 (10.3s)	8.09 (10.3s)	8.04 (1.2s)
	1,280	8.00 (10.3s)	8.00 (10.3s)	7.98 (12s)
	12,800	7.95 (31s)	7.95 (31s)	7.92 (120s)
	128,000	7.92 (265s)	7.91 (265s)	7.87 (1200s)
	1,280,000	7.89 (2640s)	7.88 (2640s)	7.83 (12000s)

Experiment

Generalization: KnapSack example

Given a set of n items $i = 1, \ldots n$, each with weight $w_i$and value v_i and a maximum weight capacity of W, the 0-1 KnapSack problem consists in maximizing the sum of the values of items present in the knapsack so that the sum of the weights is less than or equal to the knapsack capacity:

$$\max_{S \subseteq \{1,2,\ldots,n\}} \sum_{i \in S} v_i$$

$$subject\ to \sum_{i \in S} w_i \leq W$$

Experiment

Generalization: KnapSack example

Table 5: Results of RL pretraining-Greedy and Active Search on KnapSack (higher is better).

Task	RL pretraining greedy	Active Search	Random Search	Greedy	Optimal
KNAP50	19.86	**20.07**	17.91	19.24	**20.07**
KNAP100	40.27	**40.50**	33.23	38.53	**40.50**
KNAP200	57.10	**57.45**	35.95	55.42	**57.45**

Conclusion

- This paper constructs Neural Combinatorial Optimization, a framework to tackle combinatorial optimization with reinforcement learning and neural networks.
- We focus on the traveling salesman problem (TSP) and present a set of results for each variation of the framework
- The experiment shows that Neural Combinatorial Optimization achieves close to optimal results on 2D Euclidean graphs with up to 100 nodes.
- Reinforcement learning and neural networks are successful tools to solve combinatorial optimization problems if properly constructed.

Future works

- The above framework works very well when the problems are of sequence to sequence type
- Try to solve other kinds of combinatorial optimization problems using reinforcement learning

UNIVERSITY OF WATERLOO

THANK YOU!