Solving Combinatorial Optimization problems with Neural Nets

Hao Sun

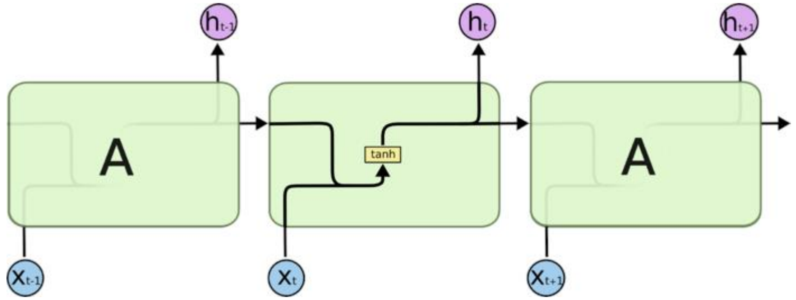
Apr.12, 2019

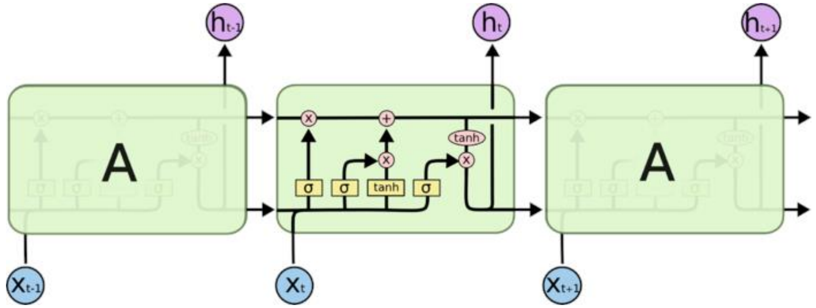
Outline

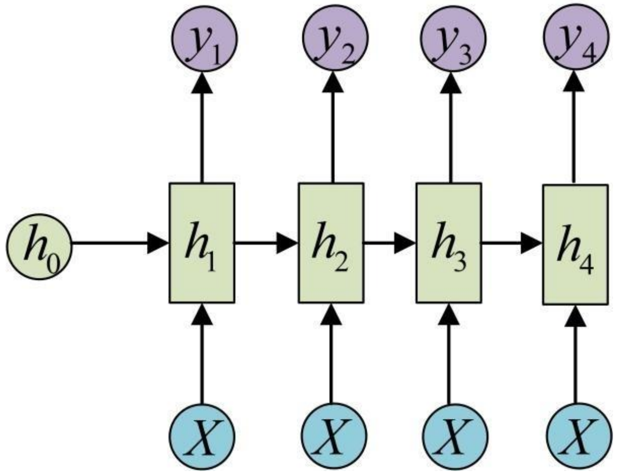
- From RNN, LSTM to Seq2Seq Models
- Pointer Networks
- Neural Combinatorial Optimization with RL
- Other extensions

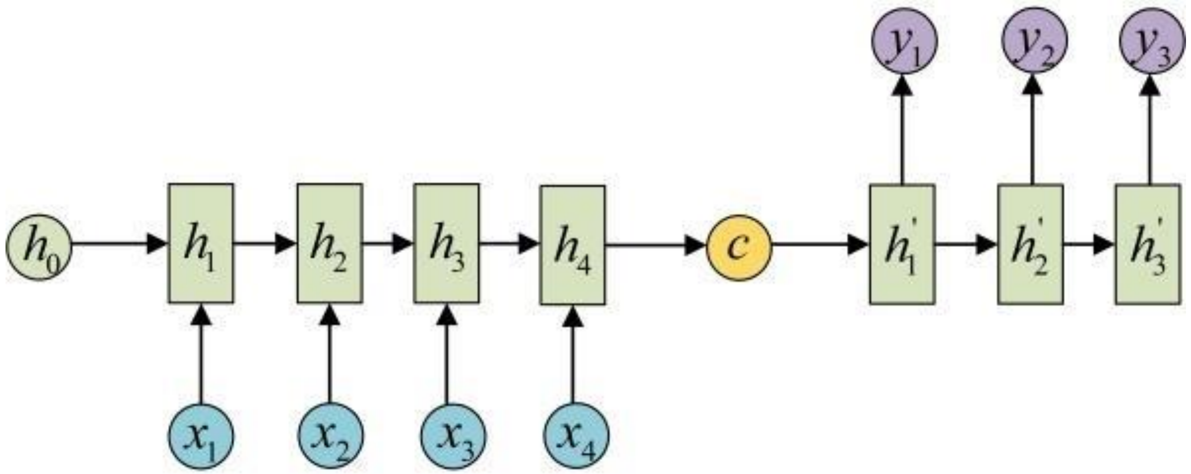
RNNs & LSTMs

- Recurrent Neural Nets
- seq2seq



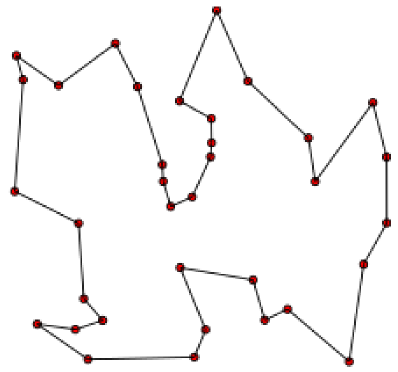


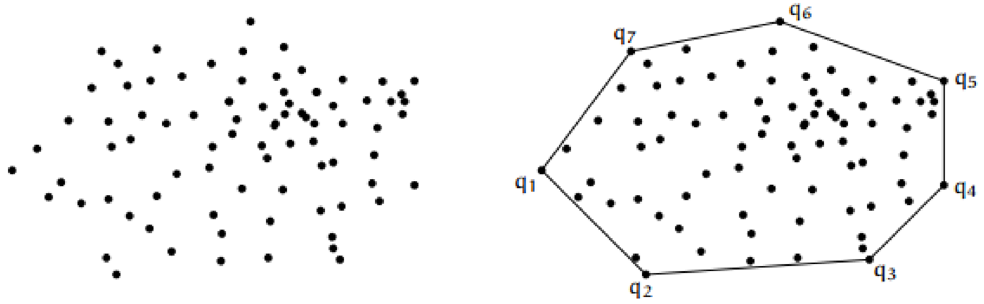




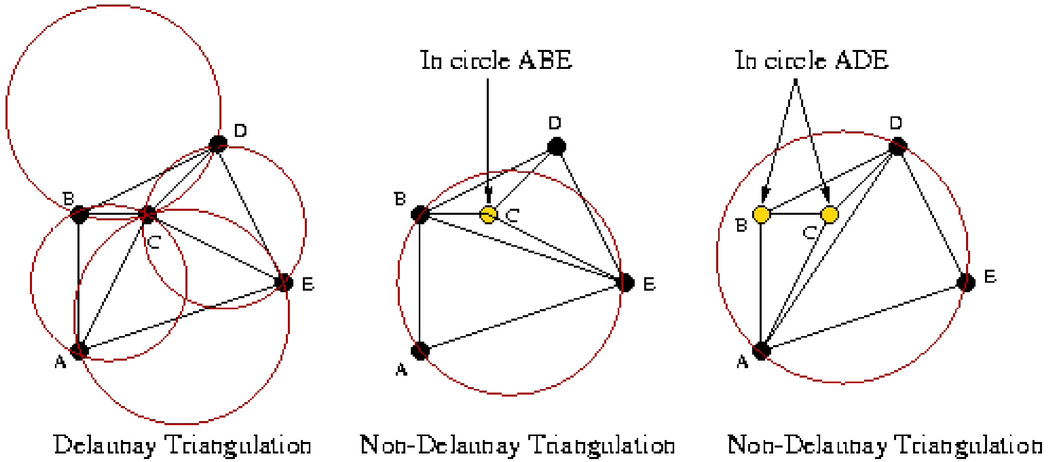
Combinatorial Optimization Problems

- Travelling salesman problem(TSP)
- Convex Hull
- Delaunay Triangulation
- P/NP

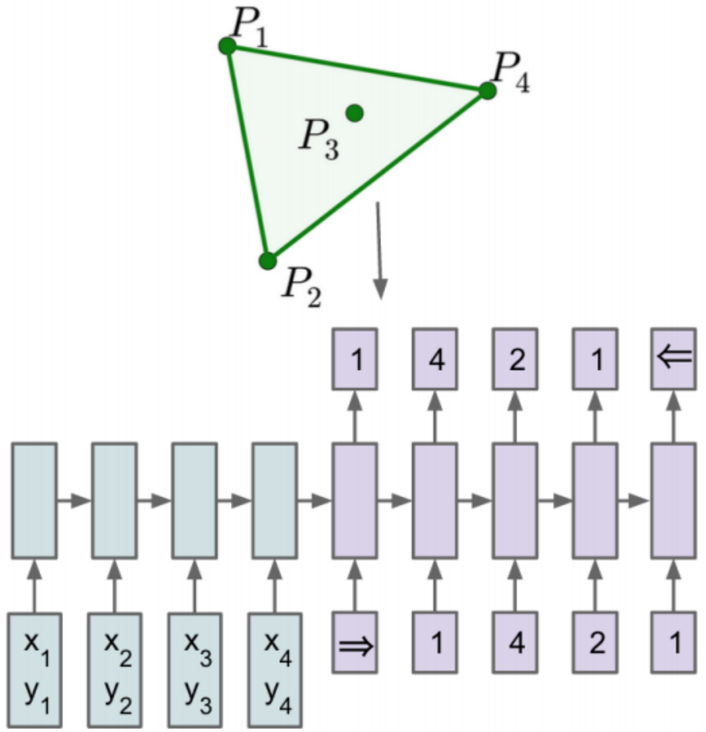




(a) Input. (b) Output.



Solving CO by NNs



- Seq2Seq
- Training pair (P, C^P)
  - $P = \{P_1, \dots, P_N\}, C^P = \{C_1, \dots, C_{m(P)}\}$
- $p(C^P | P; \theta)$ to estimate the conditional prob

- Optimization:

$$\theta^* = arg \max_{\theta} \left(\sum_{P, C^P} \log\big(p(C^P | P; \theta)\big) \right)$$

Attention module

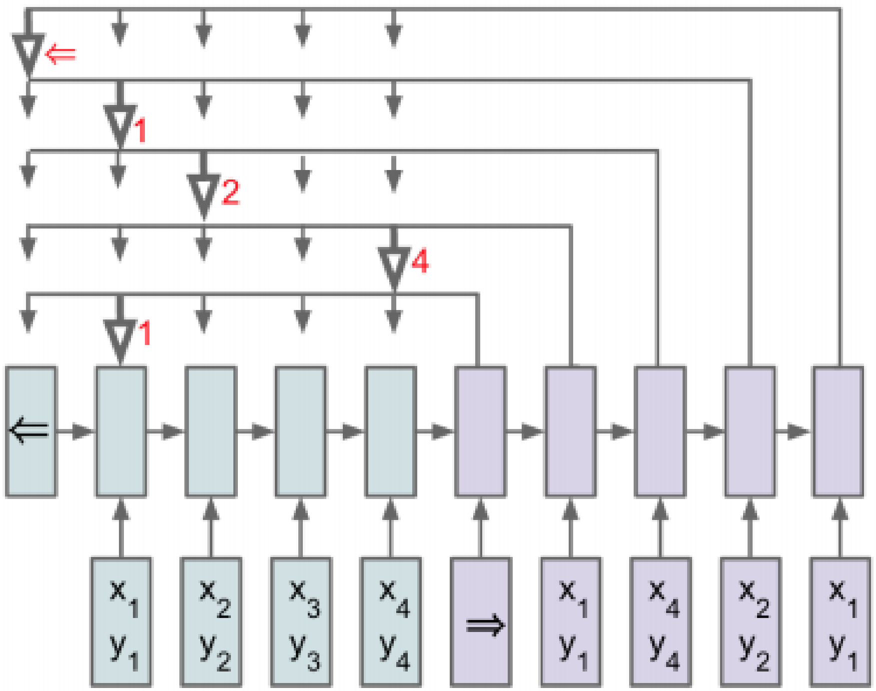
- Vanilla Seq2Seq produce C^P using the fixed dimensional state
- Constrains the amount of information and computation
- Attention:
  - Hidden states of encoder and decoder
    - $e = \{e_1, \dots, e_n\}, d = \{d_1, \dots, d_{m(P)}\}$
  - Compute attention vector by
    - $u_j^i = v^T \tanh(W_1 e_j + W_2 d_i), j \in (1, \dots, n)$
    - $a_j^i = softmax(u_j^i), j \in (1, \dots, n)$
    - $d_i' = \sum_{j=1}^{n} a_j^i e_j$
- *Not applicable to problems where the output dictionary size depends on the input.*

Pointer Networks(PN)

- Need
  - output dictionary size depends on the input

$\Longleftrightarrow$ $i.e.$

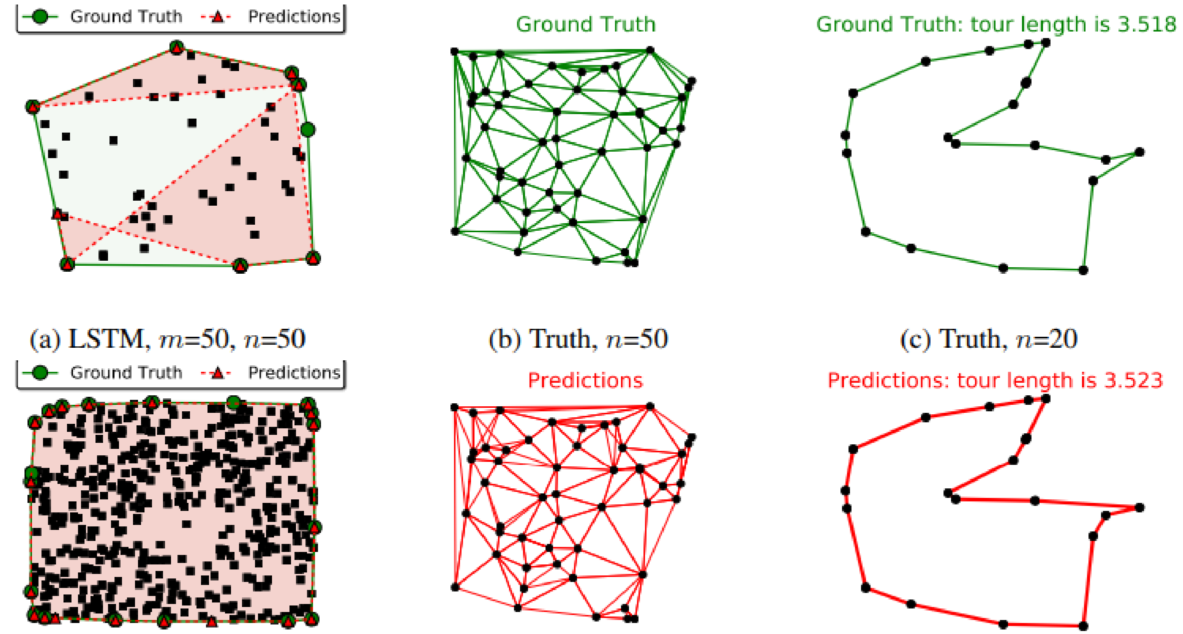
- PN
  - outputs are in a subset of inputs
    - $u_j^i = v^T \tanh(W_1 e_j + W_2 d_i)\,, j \in (1, \dots, n)$
    - $p(C_i | C_1, \dots, C_{i-1}, P) = softmax(u^i)$



Empirical results

- Table: on the convex hull problem

METHOD	TRAINED n	n	ACCURACY	AREA
LSTM [1]	50	50	1.9%	FAIL
+ATTENTION [5]	50	50	38.9%	99.7%
PTR-NET	50	50	72.6%	99.9%
LSTM [1]	5	5	87.7%	99.6%
PTR-NET	5-50	5	92.0%	99.6%
LSTM [1]	10	10	29.9%	FAIL
PTR-NET	5-50	10	87.0%	99.8%
PTR-NET	5-50	50	69.6%	99.9%
PTR-NET	5-50	100	50.3%	99.9%
PTR-NET	5-50	200	22.1%	99.9%
PTR-NET	5-50	500	1.3%	99.2%



(a) LSTM, m=50, n=50 (b) Truth, n=50 (c) Truth, n=20
(d) Ptr-Net, m=5-50, n=500 (e) Ptr-Net , m=50, n=50 (f) Ptr-Net , m=5-20, n=20

Neural Combinatorial Optimization with RL

- RL is better than SL in solving CO problems for:
  - High cost in labeling
  - Readily to define rewards
  - Generalization ability
- NCO-RL uses pointer network as its policy network and leverages policy gradient methods to train its parameters
  - REINFORCE
    - $\nabla_\theta J(\theta) \approx \frac{1}{B}\sum_{i=1}^{B}\big(L(\pi_i \mid s_i) - b(s_i)\big)\nabla_\theta \log p_\theta(\pi_i | s_i)$
  - Actor-Critic
    - $b(s_i) = b_{\theta_v}(s_i)$
  - Active Search

Extensions

- Reinforcement learning for solving vehicle routing problem
  - Non-sequence embedding
- Learning Combinatorial Optimization Algorithms over Graphs
  - Graph embedding
- Code:
  - PointNet:https://github.com/LinyangHe/PyTorch-Pointer-Network-for-Number-Sorting
  - NCO-RL:https://github.com/higgsfield/np-hard-deep-reinforcement-learning
  - Graph:https://github.com/Hanjun-Dai/graph_comb_opt

References

- Vinyals, Oriol, Meire Fortunato, and Navdeep Jaitly. "Pointer networks." *Advances in Neural Information Processing Systems*. 2015.
- Bello, Irwan, et al. "Neural combinatorial optimization with reinforcement learning." *arXiv preprint arXiv:1611.09940*(2016).
- Nazari, Mohammadreza, et al. "Reinforcement learning for solving the vehicle routing problem." *Advances in Neural Information Processing*
- Khalil, Elias, et al. "Learning combinatorial optimization algorithms over graphs." *Advances in Neural Information Processing Systems*. 2017.
- https://www.zhihu.com/question/43610101