Boosting Dynamic Programming with Neural Networks for Solving NP-hard Problems

Feidiao Yang

Institute of Computing Technology, Chinese Academy of Sciences

April 3, 2018

Outline

1. Revisiting Dynamic Programming
2. Approximating Dynamic Programming with Neural Networks
3. Application to Traveling Salesman Problem (TSP)
4. Experimental Results
5. Summary

Outline

1. Revisiting Dynamic Programming
2. Approximating Dynamic Programming with Neural Networks
3. Application to Traveling Salesman Problem (TSP)
4. Experimental Results
5. Summary

Dynamic Programming

- Dynamic programming is a powerful method for solving combinatorial optimization problems by utilizing the properties of
  - optimal substructures;
  - overlapping subproblems.

Problems	**Brute-force**	**DP**
chain matrix multiplication	Catalan number	$O(n^3)$
knapsack	$O(2^n)$	$O(cn)$
TSP	$O(n!)$	$O(2^n n^2)$

Dynamic Programming

- Multi-step decision-making problem: $s \to a \to \delta(s,a)$
  - s: the current state (subproblem)
  - a: a feasible decision under state s
  - $\delta(s,a)$: the set of subsequent sub-states
- Dynamic programming function: $f : S \to \mathbb{R}$
- Optimal substructures: $f(s) = \min_{a} \left\{ v(a) + \sum_{s' \in \delta(s,a)} f(s') \right\}$
  - $v(a)$ is the cost of making decision a
- Overlapping subproblems: tabular method
- Solution construction: $a = \arg\min_{a'} \left\{ v(a') + \sum_{s' \in \delta(s,a')} f(s') \right\}$

Challenges of DP for Solving NP-Hard Problems

- Exponential number of states
  - Exponential space complexity for tabular method
  - Exponential time to visit all states

Outline

1. Revisiting Dynamic Programming
2. Approximating Dynamic Programming with Neural Networks
3. Application to Traveling Salesman Problem (TSP)
4. Experimental Results
5. Summary

Some Related Work

- *Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly*. **Pointer networks**. NIPS 2015
  - Sequence to sequence learning with RNN
  - Solving convex hull, Delaunay triangulation, and TSP
- *Anton Milan, S. Rezatofighi, Ravi Garg, Anthony Dick, and Ian Reid*. **Data-driven approximations to np-hard problems**. AAAI 2017
  - Introducing the original objects
  - Solving quadratic assignment problem and TSP
- . . .

- Prop.
  - End-to-end learning with powerful machine learning techniques;
  - The result model can be applied to different problem instances (with some limitations).
- Cons.
  - Did not utilize the intrinsic properties of the problems;
  - Cost to make label data;
  - Can only learn approximated solution.

Representing DP Functions with Neural Networks

Using a neural network to represent a function is much more flexible than the rigid tabular method.

- Extend a DP function to a continuous function;
- Universal approximation theorem;
- A DP function does not need to be absolutely precise;
- Not all states are equally useful;
- A suboptimal solution is adequate in practice.

But it is impractical to train a neural network with the idea of supervised learning for it is even difficult to compute a single label data.

Objective of Training the Network

$$\min J(\theta) = \sum_{s \in \mathcal{S}} \left(f(s;\theta) - \min_a \left\{ v(a) + \sum_{s' \in \delta(s,a)} f(s';\theta) \right\} \right)^2 .$$

Training the model with gradient descent

$$\begin{aligned} \nabla J = & 2 \sum_{s \in \mathcal{S}} \left(f(s;\theta) - \min_a \left\{ v(a) + \sum_{s' \in \delta(s,a)} f(s';\theta) \right\} \right) \\ & \left(\nabla f(s;\theta) - \nabla \min_a \left\{ v(a) + \sum_{s' \in \delta(s,a)} f(s';\theta) \right\} \right) . \end{aligned}$$

$$\theta_{t+1} = \theta_t - \eta_t \nabla J(\theta_t).$$

Training Algorithm

Algorithm 1 Training the neural network with solution reconstruction

Initialize neural network $f(s;\theta)$ with model parameter θ, maybe with pre-training on edge cases
Initialize data pool D
Initialize exploration parameter $\varepsilon_t = 1.0$ and learning rate η_t appropriately
for each iteration $t = 1, 2, \ldots$ **do**
    Initialize state list S, state scheduling data structure Q, and state visit-marking data structure V
    Add the state s_0 representing the original problem to Q and V
    while Q is not empty **do**
        $s =$ Q.pop()
        S.Add(s)
        With probability ε_t:
            select a random feasible decision a
        with probability $1 - \varepsilon_t$:
            select decision a according to equation (**??**)
        for each sub-state $s' \in \delta(s, a)$ **do**
            if cannot find s' in V **then**
                Add s' to Q and V
            end if
        end for
    end while
    Generate a mini-batch training data B=S+Sample(D)
    Preform a gradient descent step on data B to close the gap according to equations (**??**) and (**??**)
    Add S to D with a weight being the value of the solution
    Decay ε_t and η_t if necessary
end for

Outline

1. Revisiting Dynamic Programming
2. Approximating Dynamic Programming with Neural Networks
3. Application to Traveling Salesman Problem (TSP)
4. Experimental Results
5. Summary

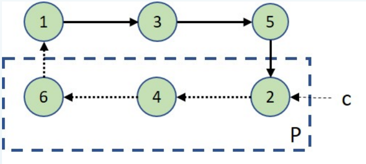
Application to TSP

Dynamic programming (Held-Karp algorithm) for solving TSP:

$$f(P, c) = \min_{c' \in P; c' \neq c} \{d(c, c') + f(P \backslash \{c\}, c')\}$$

$$c_{i+1} = \arg\min_{c \notin \{c_1, \ldots, c_i\}} \{d(c_i, c) + f([n] \backslash \{c_1, \ldots, c_i\}, c)\}$$

- $P \subseteq [n]$: vertices to be visited
- c: current starting point
- $f(P, c)$: the shortest length of paths visiting P, starting from c and returning to vertex 1



Outline

1 Revisiting Dynamic Programming

2 Approximating Dynamic Programming with Neural Networks

3 Application to Traveling Salesman Problem (TSP)

4 Experimental Results

5 Summary

Experimental Results

Table: Experimental results in TSPLIB[1]

Data	Opt.	NNDP		Held-Karp		Christofides		Greedy	
Gr17	2085	**2085**	**1**	**2085**	**1**	2287	1.1607	2178	1.0446
Bayg29	1610	**1610**	**1**	NA	NA	1737	1.0789	1935	1.2019
Dantzig42	699	**709**	**1.0143**	NA	NA	966	1.382	863	1.2346
HK48	11461	**11539**	**1.0068**	NA	NA	13182	1.1502	12137	1.059
Att48	10628	**10868**	**1.0226**	NA	NA	15321	1.4416	12012	1.1302
Eil76	538	**585**	**1.0874**	NA	NA	651	1.1128	598	1.1115
Rat99	1211	**1409**	**1.1635**	NA	NA	1665	1.3749	1443	1.1916
Br17	39	**39**	**1**	**39**	**1**	NA	NA	56	1.435
Ftv33	1286	**1324**	**1.0295**	NA	NA	NA	NA	1589	1.2002
Ft53	6905	**7343**	**1.0634**	NA	NA	NA	NA	8584	1.169

[1] The first 7 cases are symmetric TSP instances and the last 3 cases are asymmetric TSP instances. Each instance was run with 10000 iterations. The time complexity is $O(n^4)$ for each iteration, varying from 0.01 section to 0.1 sections for each iteration.

A Case Study

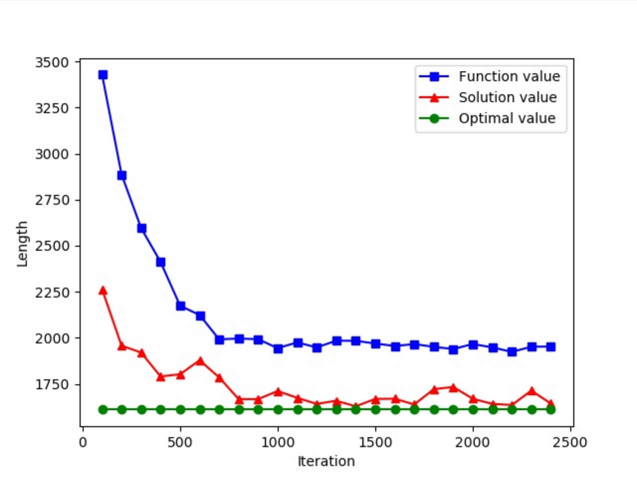


Figure: Convergence of the case Bayg29. The function values are the values the model outputs; the solution values are the exact values of the solutions constructed from the model.

Outline

1. Revisiting Dynamic Programming
2. Approximating Dynamic Programming with Neural Networks
3. Application to Traveling Salesman Problem (TSP)
4. Experimental Results
5. Summary

Summary

- An approach to boost the capability of dynamic programming with neural networks.
  - tabular method $\rightarrow$ neural network (polynomial size), approximately representing DP functions;
  - iterative training algorithm with data generated from a solution reconstruction process;
  - approximating ability and flexibility of neural networks + the advantage of dynamic programming;
  - significantly reduce the required space for some NP-hard problems, trading-off space, running time, and accuracy.
- Apply to the Held-Karp algorithm for TSP.
  - can handle larger problems that are intractable for the conventional DP;
  - outperform other well know approximation algorithms.

Thank You!