Learning Classic Online Optimization Algorithms via RL

Aranyak Mehta
Market Algorithms,
Google Research, Mountain View
Paper: A new dog learns old tricks: RL finds classic optimization algorithms

To Appear in International Conference on Learning Representations (ICLR), 2019

Collaborators:

Chris Liaw
University of British Columbia

William Kong
Georgia Tech

D Sivakumar
Google Research
The Undeniable Success of Deep Learning

Basic Cognition Tasks:
Object Recognition

Basic Language Tasks:
Question-Answering in short paragraphs

Finite planning Tasks:
Atari, Chess, Go, ...

Some work on combinatorial optimization as well (our focus)

Bello et al. 2016,
Dai et al. 2017,
Kool-Welling 2018,
Boutilier-Lu 2016,...
Can machine learning solve classic algorithms problems?
Can machine learning solve online optimization problems?
CAN DEEP REINFORCEMENT LEARNING SOLVE ONLINE OPTIMIZATION PROBLEMS?
CAN DEEP REINFORCEMENT LEARNING DESIGN WORST CASE, UNIFORM ONLINE OPTIMIZATION ALGORITHMS?
Outline

1. Describe three problems and algorithms
2. Describe the RL setup
3. Universal Training Distribution
4. Results
Three Online Optimization Problems

1. Adwords
2. Knapsack
3. Secretary / Optimal Stopping
Problem 1: The AdWords Problem

**Definition** (M., Saberi, Vazirani, Vazirani, FOCS 2005, JACM 2007)

*N* advertisers, advertiser *a* has budget *B(a)*

*M* search queries that arrive online, advertiser *a* has bid *v(a, q)* for query *q*

Algorithm needs to allocate *q* to zero or one of the advertisers irrevocably. Allocated advertiser depletes budget by *v(a, q)*.

**Goal:** Maximize sum of values over all allocations

Generalizes online bipartite matching
Special case where values are in \{0, 1\} is called (online) budgeted or b-Matching

Very impactful abstraction and algorithmic technique in industry.
The MSVV Algorithm

For advertiser $a$, define $\text{spent}(a) = \text{fraction of their budget already exhausted.}$

When query $q$ arrives, allocate it to an advertiser that maximizes

$$\text{bid}(a, q) \times \Psi(\text{spent}(a))$$

where

$$\Psi(x) = 1 - \exp(-(1 - x)).$$
The MSVV Algorithm

For advertiser $a$, define $spent(a) = \text{fraction of their budget already exhausted}$. When query $q$ arrives, allocate it to an advertiser that maximizes

$$bid(a, q) * \Psi(spent(a))$$

where

$$\Psi(x) = 1 - \exp(-(1 - x))$$

Theorem [MSVV05]
Achieves optimal competitive ratio $1 - 1/e$

Note: the curve is the best estimate of the optimal dual variable.
Problem 2: Online Knapsack (with small sizes)

Knapsack with size B. Items arrive over time.

Item(t) has: value(t), size(t).

Decision: Take the item in the knapsack, or not.

Goal: Maximize sum_value s.t., sum_size <= B.

We will mostly consider the iid version: (value(t), size(t)) picked iid from some known distribution F(v, s).

Assume all sizes are small compared to B.
Online Knapsack Algorithm

Algorithm "Bang-per-Buck": Pick item $t$ iff $\frac{\text{value}(t)}{\text{size}(t)} \geq T^*$

where $T^*$ depends on the distribution $F$.

**Note:** $T^*$ is the optimal value of the dual variable.
Problem 3: Secretary / Optimal Stopping

Items arrive one by one. Item(t) has a value(t).

Pick one item to maximize value
  i.e., Maximize Prob[picking the item with highest value].

- **Adversarial version: [Dynkin ‘63]**
  - No constraint on the values, but items arrive in a *random order*.
  - How to represent worst case numbers: define a binary / percentile setting.
  - Algorithm “Wait-then-Pick”: Discard the first $1/e$ fraction of the items; then pick any item better than the max so far.

- **iid version [Gilbert Mosteller ‘66]:**
  - Item values picked iid from some known distribution $F$.
  - Algorithm “Decreasing Thresholds”: Sequence of decreasing thresholds $T(1) \geq T(2) \geq .. \geq T(n)$

**Note:** Again, the algorithms have a primal-dual interpretation [Buchbinder et al. 2014]
Why these problems?

Simple formulations; concise algorithms.

- Natural ideas like Greedy do not work; Algorithms are simple yet subtle. Very different from each other
- Nice structure for testing input length independence (AdWords)
- AdWords useful: If RL works, we can hope that it will adapt nicely to input distributions that arise in practice

All have natural primal-dual interpretation (Buchbinder, Jain, Naor ‘07)

- If RL works, we can hope to solve a larger class of problems
- We can test the choices made by the RL algorithm to see if it “mimics” the primal-dual algorithm
RL setting

1. Each problem has a natural MDP representation
   - No hints towards algorithms!

2. Standard learning networks and training algorithms

3. “Adversarial” training sets
   - Universal and High-entropy
AdWords MDP

States:
When query arrives, agent sees:
- bid of each advertiser for the query
- fraction of budget spent for each advertiser (discretized)

Actions: Choice of advertiser to allocate query to, OR “Not-Allocated”

Reward: bid of the chosen advertiser if she has budget left, 0 otherwise

Transition:
- fraction of budget spent increased by bid / Budget for chosen advertiser;
- next query generated and a new state vector produced
Learning an Agent

Goal: Learn agent’s policy function that maps vector of (value, fractional spend) pairs to index of an advertiser (or n/a)

Train a 5-layer 500-neuron-per-layer network with ReLU non-linearity

Standard REINFORCE policy-gradient learning with learning rate $1e^{-4}$, batch size 10, bootstrap over input stream size.

Takes few hours typically on single-threaded standard Linux desktop
MDP for Knapsack and Secretary

Basic versions are straightforward:

- **Knapsack:** \( (v(i), s(i), i/n, \text{Spent}(i)/\text{Budget}) \)
- **Secretary:** \( (v(i), i/n) \)
  - Define a
  - Reward given at the end-state
  - Not an MDP (rewards not Markovian). Can be converted to MDP, but still works as is!

These work for some settings

- Knapsack: iid
- Secretary: iid, binary / percentile.

For truly worst case algorithms, need to augment the state by some “memory”.

- A data structure that RL has to learn to maintain (e.g., histogram of bang-per-buck,
Adversarial Training Sets
Goal

“Right way” to do it is “YaoGAN” (not in paper, currently active WIP):

Set up a zero-sum game between two agents:

- a generator $G$ that generates “hard” instances
- algorithm $A$ that tries to perform well on those instances
Go back to the algorithms papers!

A typical paper:

- **Sections 1--4: Problem, Algorithm, and Analysis**
  - Algorithm gets at least $1-1/e$ factor of OPT.

- **Section 5: Upper Bound**
  - No Algorithm can get better than $1-1/e$
  - Obtained via carefully crafted distributions.
  - Yao's Lemma: automatically gives a bound on Randomized Algorithms as well.

**Idea:**
- These distributions capture all the hardness of the problems. Use them as training sets -- *Universal Distributions*.
- If not available for the problem at hand, find *High-Entropy Distributions*.
Universal Distribution for 0/1 Adwords
Universal Distribution for 0/1 Adwords
Universal Distribution for 0/1 Adwords

Random also does well on upper-triangular, so we add “thick-Z” into the distribution as well.
Results
Table 4: Comparison of the MSVV algorithm and the learned algorithm with discretized state space.

<table>
<thead>
<tr>
<th>Distribution</th>
<th>No. of advertisers</th>
<th>No. of ad slots</th>
<th>MSVV</th>
<th>Learned</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adversarial graph, ad slots permuted, values in {0, 1}</td>
<td>5</td>
<td>50</td>
<td>45.7 ± 0.1</td>
<td>45.1 ± 0.1</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>100</td>
<td>91.0 ± 0.2</td>
<td>91.4 ± 0.3</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>500</td>
<td>482.3 ± 0.3</td>
<td>462.6 ± 1.4</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1000</td>
<td>976.0 ± 0.6</td>
<td>930.0 ± 2.7</td>
</tr>
<tr>
<td>Adversarial graph, ad slots permuted, values drawn randomly</td>
<td>5</td>
<td>50</td>
<td>23.1 ± 0.7</td>
<td>22.4 ± 0.7</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>100</td>
<td>45.4 ± 1.0</td>
<td>46.3 ± 1.0</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>500</td>
<td>238.4 ± 5.7</td>
<td>240.0 ± 6.1</td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>1000</td>
<td>471.4 ± 7.5</td>
<td>474.2 ± 7.4</td>
</tr>
</tbody>
</table>

Trained only on inputs of length 100
How does the network solve it?

Did it “Find the MSVV Algorithm”? How to evaluate?

Since the network is succinct, it has to represent a succinct logic...

1. Probing the network as a black box.

   **Warm-up: 0/1 bids**

   Pretend we’re in the middle of execution for an instance. We’re at an item arrival.

   All advertisers have bid=1
   All except advertiser i have spend=0.5.

   *x-axis:* spend
   *y-axis:* Probability that advertiser i wins the item
How does the network solve it?

Did it “Find the MSVV Algorithm”? How to evaluate?

1. Probing the network as a black box.

General Case:

All advertisers except advertiser 0 have bid=1, spend=0.5.

x-axis: spend(0)
y-axis: Minimum bid to win the item.

Blue: Learned Agent
Green: OPT (MSVV)
2. Measure how “duals” evolve. [Ack: ICLR reviewer suggestion]

How do the duals evolve?

“Worst case” upper-triangular graph.

Blue: Duals for MSVV / Balance Algorithm.

Red: Imputed duals for the Learned agent.

Different plots for different offline vertices (advertisers).
Truly Input-Length Independent?

MSVV is a simple “policy” that can be described succinctly for arbitrary number of advertisers and query stream length.

Idea: Discretize the space further!

When query $q$ arrives, agent sees a 100x100 state vector $S$, where $S[c, d] =$ fraction of advertisers with value $c$ for $q$ and $d\%$ of budget spent.

Action space: agent chooses $c, d \in [100]$.

Query allocated to a random agent with (value, spent fraction) in the $(c, d)$ bucket if that bucket is non-empty.
It works!

Table 3: This table compares the performance of the learned algorithm compared the BALANCE in the discretized state space. Here, the agent is trained on the adversarial graph with the ad slots arriving in a permuted order. The agent was only trained on the input instance with 20 advertisers and a common budget of 20 but tested on instances with up to $10^6$ ad slots.

<table>
<thead>
<tr>
<th>No. of advertisers</th>
<th>Budgets (common)</th>
<th>No. of ad slots</th>
<th>Approx. of BALANCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>10</td>
<td>100</td>
<td>0.9</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>400</td>
<td>0.92</td>
</tr>
<tr>
<td>30</td>
<td>30</td>
<td>900</td>
<td>0.88</td>
</tr>
<tr>
<td>10</td>
<td>2000</td>
<td>20000</td>
<td>0.85</td>
</tr>
<tr>
<td>10</td>
<td>4000</td>
<td>40000</td>
<td>0.85</td>
</tr>
<tr>
<td>25</td>
<td>4000</td>
<td>100000</td>
<td>0.84</td>
</tr>
<tr>
<td>50</td>
<td>400</td>
<td>20000</td>
<td>0.84</td>
</tr>
<tr>
<td>100</td>
<td>100</td>
<td>10000</td>
<td>0.85</td>
</tr>
<tr>
<td>100</td>
<td>1000</td>
<td>100000</td>
<td>0.85</td>
</tr>
<tr>
<td>200</td>
<td>100</td>
<td>20000</td>
<td>0.85</td>
</tr>
<tr>
<td>500</td>
<td>50</td>
<td>25000</td>
<td>0.85</td>
</tr>
<tr>
<td>1000</td>
<td>100</td>
<td>10000</td>
<td>0.84</td>
</tr>
<tr>
<td>25</td>
<td>40000</td>
<td>1000000</td>
<td>0.84</td>
</tr>
</tbody>
</table>
Other restricted models

Agent performs optimally in restricted input models as well, suggesting it finds the optimal algorithms for those settings:

- Bounded-degree graphs
  - Greedy gets ~1 [Naor Wajc '14]
- Arbitrary graph, 0/1 bids, random arrival order
  - Balance gets ~1 [Motwani Panigrahi Xu '06]
- Power law graphs
Knapsack

**IID Setting:** (value, size) \(\leftarrow U[0,1] \times U[0,1]\)

\((20, 200)\)  \hspace{1cm}  \((50, 500)\)  \hspace{1cm}  \((50, 1000)\)

- **x-axis:** value / size
- \(\leftarrow (\text{Budget}, \text{num\_items})\)
- \(\leftarrow \text{Prob[accept]}\)
- \(\leftarrow \text{Histogram of accept vs all}\)

Achieves \(\sim 96\%\) of OPT (w/discretization)
**Knapsack**

- **Adversarial setting**
  - MDP has to be augmented by “memory” of the histogram of bang-per-buck seen so far.
  - *No known Adversarial Distribution* in the literature.
  - Open question. First step:

  ![Diagram](diagram.png)

  - Single threshold does not work.
    - 73% of OPT
  - Augment the state by a histogram.
    - 96% of OPT.
    - Cheating (hist of val/size)
  - WIP/Open: “memory” version without the cheating.
Secretary: Binary / Percentile setting

- **Adversarial input**
  - **Issue**: is numbers have to be unbounded for truly adversarial input
  - **Solution**: Input is a random permutation of $[1..n]$, Agent sees only through a filter:
    - **Binary**: is the item the max so far?
    - **Percentile**: what is the item’s percentile so far?

Red: **Optimal Algorithm**
Recall wait until $\sim 0.37n$ and pick next item with percentile 100

Blue: **Learned Agent if it sees percentile 100**

Green: **Learned Agent if it sees percentile 95**
Secretary IID

- Item values iid from $U[0,1]$
- Recall OPT is the *Decreasing Thresholds Algorithm*
- Agent finds something similar
  - Different curves for different input lengths.
Conclusions

Wanted to learn algorithms, and we did!

Three key ideas/results

- architecture sweet spot: online optimization problems for symmetric problems
- use of TCS-style “hard distributions” on training instances leads to robust algorithms
- black-box analysis of learned algorithms reveals behaviors of classic algorithms