Combining Algorithms and NNs

SEMINAR IN DEEP IN DEEP NEURAL NETWORKS

Kadoglou Maria Eleni, 18.04.21
Exact Combinatorial Optimization with Graph Convolutional Neural Networks
Branch & Bound Algorithm

Mixed-Integer Linear Program (MILP)

\[
\begin{align*}
\text{arg min}_x & \quad c^T x \\
\text{subject to} & \quad Ax \leq b, \\
& \quad l \leq x \leq u, \\
& \quad x \in \mathbb{Z}^p \times \mathbb{R}^{n-p}
\end{align*}
\]

NP-Hard Problem

Linear Program Relaxation

\[
\begin{align*}
\text{arg min}_x & \quad c^T x \\
\text{subject to} & \quad Ax \leq b, \\
& \quad l \leq x \leq u, \\
& \quad x \in \mathbb{R}^n
\end{align*}
\]

Branch & Bound Algorithm

Convex Problem \rightarrow \text{lower bound to the original MILP}
Branch & Bound Algorithm

- Let $x^* \in \mathbb{Z}^p \times \mathbb{R}^{n-p} \rightarrow$ solution to the original problem (lucky!)
- Let $x^* \notin \mathbb{Z}^p \times \mathbb{R}^{n-p} \rightarrow$ decompose into two sub-problems

1. Pick a fractional variable to branch on

Obj = 2

Obj = 4

x1,x2 fractional

Obj = 5

x1,x2 fractional

• **Lower Bound**: minimum of leaf nodes

• **Upper Bound**: minimum of leaf nodes with integer solution

2. Continue branching to the fractional variables

Obj = 5

x1,x2 fractional

Obj = 6.5

x1,x2 fractional

Obj = 8

x1,x2 integral

Obj = 5.5

x1,x2 fractional

Obj = 7

x1,x2 integral

3. The process stops:

- $\text{LB} = \text{UB}$
- $\text{LB} - \text{UB} = \varepsilon$
- The feasible regions do not decompose anymore

- Branching selection

- Node Selection
Branch & Bound Algorithm

Fundamental Questions: Which variable to choose for branching?

- SOTA: Strong Branching
- Proposed Method: The Neural Network will say

\[ x_1 \leq [x_1^*] \quad x_1 \geq [x_1^*] \quad x_2 \leq [x_2^*] \quad x_2 \geq [x_2^*] \]

\[ x = x_1^* \quad x = x_2^* \]

1. Collect expert state-action pairs \( D = \{(s_i, a_i^*)\}_{i=1}^{N} \)
2. Learn the policy by minimizing:

\[ \mathcal{L}(\theta) = -\frac{1}{N} \sum_{(s,a) \in D} \log \pi_\theta(a^* | s) \]
Proposed Method

How to represent the states?

Growing structure → Graph

Original problem, bounds

Relaxation problem, solution to LPs, added constraints

1. **State Encoding** → bipartite graphs with attributes

Mixed-Integer Linear Program (MILP)

\[
\text{arg min}_x c^T x \\
\text{subject to } Ax \leq b, \\
l \leq x \leq u, \\
x \in \mathbb{Z}^p \times \mathbb{R}^{n-p}
\]

\[s_t = (G, C, E, V)\]
Proposed Method

How can we model the Graph to the Neural Network?

2. **Policy** $\pi_\theta(a|s_t) \rightarrow$ Graph Convolutional Neural Network (GCNN)

$$c_i \leftarrow f_c(c_i, \sum_{j}^{(i,j) \in \mathcal{E}} g_c(c_i, v_j, e_{i,j})) \quad v_j \leftarrow f_v(v_j, \sum_{i}^{(i,j) \in \mathcal{E}} g_v(c_i, v_j, e_{i,j}))$$

**Why GCNN?**

- They have permutation invariance
- Combine the node with each neighbors
- From variable to constraints
- From constraints to variables

3. Treat the problem as a **classification** one
Evaluation Results

• **Benchmarks** : 4 Np Hard Problems
• **Solver** : SCIP 6.0.1 open source solver
• **Baselines** : Hybrid Branching [RBP], Full Strong Branching Expert [FSB], SVRRANK, LMART, and the regression approach of Alvarez [TREES]

Comparing accuracy of ML models

<table>
<thead>
<tr>
<th>Set Covering</th>
<th>Combinatorial Auction</th>
<th>Capacitated Facility Location</th>
<th>Maximum Independent Set</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>acc@1</td>
<td>acc@5</td>
<td>acc@10</td>
</tr>
<tr>
<td>TREES</td>
<td>51.8±0.3</td>
<td>80.5±0.1</td>
<td>91.4±0.2</td>
</tr>
<tr>
<td>SVMRANK</td>
<td>57.6±0.2</td>
<td>84.7±0.1</td>
<td>94.0±0.1</td>
</tr>
<tr>
<td>LMART</td>
<td>57.4±0.2</td>
<td>84.5±0.1</td>
<td>93.8±0.1</td>
</tr>
<tr>
<td>GCNN</td>
<td>65.5±0.1</td>
<td>92.4±0.1</td>
<td>98.2±0.0</td>
</tr>
</tbody>
</table>
Evaluation Results

Performance regarding Solution:

Train on small (Easy) instances and evaluate generalization on medium (Medium) and large (Hard)

Results

- better in terms of solving time
- generalizes to fairly larger instances
- performance decreases as the model is evaluated on larger problems
- outside of the training distribution ??
- need data for training
How Combinatorial Optimization can help in Deep Learning?

- Problem: **differentiability** of the combinatorial components

- SOTA approaches:  
  - Solve a relaxation problem
  
- Sub-optimal:  
  - Runtime
  - Performance
  - Optimality Guarantees

e.g. Predict the quickest routes in Google Maps based on map input as an image

Construct Hybrid Architectures
Differentiation of Blackbox Combinatorial Solvers

Gradients of Blackbox Solver

Continuous Input $w$  
Discrete Output $y$

- Cost Function: $c(w, y) = w \cdot \phi(y)$
- $\omega \rightarrow$ edge weights of a graph

$\omega \mapsto y(\omega)$ such that $y(w) = \arg\min_{y \in Y} c(w, y)$
Differentiation of Blackbox Combinatorial Solvers

Shortest Path Problem from raw Images

\[ w \rightarrow \text{representation} \]

output: predicted shortest path for the respective map

\[
\begin{pmatrix}
1 & 0 & \cdots & 0 \\
1 & 0 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 1 \\
\end{pmatrix}
\]

\( k \times k \) indicator matrix of shortest path

Loss: Hamming distance between the true and the predicted SP

Piecewise function
Differentiation of Blackbox Combinatorial Solvers

Forward propagation

\[ x \rightarrow \text{NN} \rightarrow \omega \rightarrow \text{Solver} \rightarrow y \rightarrow L(y) \]

Backpropagation

\[ \frac{dL}{dx} \leftarrow \text{NN} \leftarrow \frac{dL}{d\omega} \leftarrow \text{Solver} \leftarrow \frac{dL}{dy} \leftarrow L \]

Problem!! \[ \frac{dL}{d\omega} \quad \text{Useless in Optimization} \]
Method of Interpolation

General Approaches → Relaxation → Loose a lot of information
Method of Interpolation

\[ \frac{dL}{d\omega} = \frac{dL}{dy} \frac{dy}{d\omega} \]

**We want a trick!**

**Linearization**

\[ f(y) = L(\hat{y}) + \frac{dL}{dy}(\hat{y}) \cdot (y - \hat{y}) \]

\[ \frac{df(y(w))}{dw} = \frac{dL}{dw} \]

**Interpolation**

\[ f_\lambda(w) = f(y_\lambda(w)) - \frac{1}{\lambda} \left[ c(w, y(w)) - c(w, y_\lambda(w)) \right] \]

\[ y_\lambda(w) = \text{arg min}_{y \in Y} \left\{ c(w, y) + \lambda f(y) \right\} \]

**Gradient**

\[ \nabla f_\lambda(w) = -\frac{1}{\lambda} \left[ \frac{dc}{dw}(w, y(w)) - \frac{dc}{dw}(w, y_\lambda(w)) \right] = -\frac{1}{\lambda} \left[ y(w) - y_\lambda(w) \right] \]
Experiments

Input $k \times k$ indicator matrix of shortest path

Label
\[
\begin{pmatrix}
1 & 0 & \cdots & 0 \\
1 & 0 & \cdots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \cdots & 1
\end{pmatrix}
\]

Input $k \times k$ adjacency matrix with optimal TSP tour

Label
\[
\begin{pmatrix}
0 & \cdots & 1 \\
\vdots & \ddots & \vdots \\
1 & \cdots & 0
\end{pmatrix}
\]

### Embedding Dijkstra

<table>
<thead>
<tr>
<th>$k$</th>
<th>Train %</th>
<th>Test %</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>99.7 ± 0.0</td>
<td>96.0 ± 0.3</td>
</tr>
<tr>
<td>18</td>
<td>98.9 ± 0.2</td>
<td>94.4 ± 0.2</td>
</tr>
<tr>
<td>24</td>
<td>97.8 ± 0.2</td>
<td>94.4 ± 0.6</td>
</tr>
<tr>
<td>30</td>
<td>97.4 ± 0.1</td>
<td>94.0 ± 0.3</td>
</tr>
</tbody>
</table>

### ResNet18

<table>
<thead>
<tr>
<th></th>
<th>Train %</th>
<th>Test %</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>100.0 ± 0.0</td>
<td>23.0 ± 0.3</td>
</tr>
<tr>
<td>18</td>
<td>99.9 ± 0.0</td>
<td>0.7 ± 0.3</td>
</tr>
<tr>
<td>24</td>
<td>100.0 ± 0.0</td>
<td>0.0 ± 0.0</td>
</tr>
<tr>
<td>30</td>
<td>95.6 ± 0.5</td>
<td>0.0 ± 0.0</td>
</tr>
</tbody>
</table>
The Case for Learned Index Structures

Can indexing data structures be replaced with machine learning models?

Fundamental Algorithms & Data Structure

Why ML?

- Powerful GPU → Parallelism
- Speed and Memory usage
- Benefit from data distributions
- Read-only in-memory data
B-Tree Index

Basic Idea: "Predict" the position

1. Key → Model → Pos = f(key)

2. Do Binary search inside: [Pos - min_error, Pos + max_error]

3. How to find this error?

4. Run all the keys through the model and take the maximum (over, under) miss-predictions
B-Tree Index

What eventually does the model? **“Modeling” the CDF of the key distribution**

Position = \( P(X \leq \text{key}) \times Number_{keys} \)

- So first need to learn the data distribution
- Benefit because CDFs in ML are well studied over decades

**First Attempt**

- 200M web-server log records by timestamp sorted
- 2 layer NN with 32 neurons/layer + ReLU

**Goal**: given the timestamp (index) → predict the position

+ Measure the look-up time
B-Tree Index

Result?

But why?

1. Tensorflow → designed for large models
2. B-Trees are good in overfitting
3. B-Trees → cache and operation efficient
4. Predict the region not the exact position

Need to apply a search method (e.g. binary search)
1. **Learning Index Framework (LIF)**

- Index synthesis system

  - Train the model with Tensorflow
  - Export the optimized parameter values
  - Recreate the model architecture using C++
  - Make Predictions

2. **Recursive Model Index**

   "Still Problems!"

   "Solution"

   ![Diagram showing recursive model index with stages and models]
3. Hybrid Indexes

- Build Mixtures of Models
- Worse case Scenario: B-Tree!

Results

<table>
<thead>
<tr>
<th>Type</th>
<th>Config</th>
<th>Map Data</th>
<th>Web Data</th>
<th>Log-Normal Data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Size (MB)</td>
<td>Lookup (ns)</td>
<td>Model (ns)</td>
</tr>
<tr>
<td>Btree</td>
<td>page size: 32</td>
<td>52.45 (4.00x)</td>
<td>274 (0.97x)</td>
<td>198 (72.3%)</td>
</tr>
<tr>
<td></td>
<td>page size: 64</td>
<td>26.23 (2.00x)</td>
<td>277 (0.96x)</td>
<td>172 (62.0%)</td>
</tr>
<tr>
<td></td>
<td>page size: 128</td>
<td>13.11 (1.00x)</td>
<td>265 (1.00x)</td>
<td>134 (50.8%)</td>
</tr>
<tr>
<td></td>
<td>page size: 256</td>
<td>6.56 (0.50x)</td>
<td>267 (0.99x)</td>
<td>114 (42.7%)</td>
</tr>
<tr>
<td></td>
<td>page size: 512</td>
<td>3.28 (0.25x)</td>
<td>286 (0.93x)</td>
<td>101 (35.3%)</td>
</tr>
<tr>
<td>Learned Index</td>
<td>2nd stage models: 10k</td>
<td>0.15 (0.01x)</td>
<td>98 (2.70x)</td>
<td>31 (31.6%)</td>
</tr>
<tr>
<td></td>
<td>2nd stage models: 50k</td>
<td>0.76 (0.06x)</td>
<td>85 (3.11x)</td>
<td>39 (45.9%)</td>
</tr>
<tr>
<td></td>
<td>2nd stage models: 100k</td>
<td>1.53 (0.12x)</td>
<td>82 (3.21x)</td>
<td>41 (50.2%)</td>
</tr>
<tr>
<td></td>
<td>2nd stage models: 200k</td>
<td>3.05 (0.23x)</td>
<td>86 (3.08x)</td>
<td>50 (58.1%)</td>
</tr>
</tbody>
</table>
**Hash Maps**

(a) Traditional Hash-Map

(b) Learned Hash-Map

**Goal: Reduce Conflicts**

Again Learn Distributions!

\[ h(K) = P(X \leq K) \times M \]
## Hash Maps - Results

<table>
<thead>
<tr>
<th></th>
<th>% Conflicts Hash Map</th>
<th>% Conflicts Model</th>
<th>Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map Data</td>
<td>35.3%</td>
<td>07.9%</td>
<td>77.5%</td>
</tr>
<tr>
<td>Web Data</td>
<td>35.3%</td>
<td>24.7%</td>
<td>30.0%</td>
</tr>
<tr>
<td>Log Normal</td>
<td>35.4%</td>
<td>25.9%</td>
<td>26.7%</td>
</tr>
</tbody>
</table>
**Bloom Filter**

Is this Key existed?

- **Yes**
- **No**

- False Positives
- False Negatives = 0

(a) Traditional Bloom-Filter Insertion

- key1
  - $h_1$
  - $h_2$
  - $h_3$

- key2
  - $h_1$
  - $h_2$
  - $h_3$

- key3
  - $h_1$
  - $h_2$
  - $h_3$

- ML Model??

- Bloom Filter for all misclassified records

(c) Bloom filters as a classification problem

- Key → Model → No → Bloom filter
  - Yes
  - Yes

- For the No we want to be sure!
Bloom Filter - Results

36% Memory Reduction!
The Case for Learned Index Structures – Future Work
References

• Exact Combinatorial Optimization with Graph Convolutional Neural Networks, https://arxiv.org/abs/1906.01629


• https://towardsdatascience.com/graph-convolutional-networks-deep-99d7fee5706f