Building Scalable Systems for Reinforcement Learning and Using Reinforcement Learning for Better Systems

Presented by Yuandong Tian
Research Scientist and Manager
Facebook AI Research
Overview

Building Scalable System for Reinforcement Learning (RL)

Learn Hand-tuned Heuristics by RL / ML
Building Scalable System for RL
Crash Course of Reinforcement Learning

Agent

Environment

State $s_t$

Reward $r_t$

Action $a_t$

Transition $s_{t+1}$

Reward $r_{t+1}$
Reinforcement Learning works, but expensive

<table>
<thead>
<tr>
<th>Year</th>
<th>Projects</th>
<th>Human Data</th>
<th>Training Resource</th>
<th>Training time</th>
</tr>
</thead>
<tbody>
<tr>
<td>2016</td>
<td>DeepMind’s AlphaGo</td>
<td>Yes</td>
<td>~50 GPUs + ? CPUs</td>
<td>~1 week</td>
</tr>
<tr>
<td>2017</td>
<td>DeepMind’s AlphaGo Zero (20 blocks)</td>
<td>No</td>
<td>~2000 TPUs</td>
<td>3 days</td>
</tr>
<tr>
<td>2017</td>
<td>DeepMind’s AlphaZero (20 blocks)</td>
<td>No</td>
<td>~5000 TPUs</td>
<td>8 hours</td>
</tr>
<tr>
<td>2018</td>
<td>OpenAI Five</td>
<td>No</td>
<td>128,000 CPUs + 256 GPUs</td>
<td>Several months</td>
</tr>
<tr>
<td>2019</td>
<td>DeepMind’s AlphaStar</td>
<td>Yes</td>
<td>16,000 CPUs + 3072 TPUv3 cores</td>
<td>44 days</td>
</tr>
</tbody>
</table>
Challenges in large-scale RL Training System

• Trade-offs in a *heterogenous* system
  • **Different kind of objects**: Actor / Environment / Trainer / Replay buffer
  • CPUs / GPUs Allocations
  • Multi-threading versus Multiple Processes, Batching issues
  • Local versus Distributed
  • Synchronization / Asynchronization.
    • On-policy versus off-policy methods
    • Perfect synchronization might NOT give you the best performance

• Mingled Algorithm Design and System Design
  • New System design ↔ New RL algorithm
Distributed System for training RL agent

GORILLA
[Massively Parallel Methods for Deep Reinforcement Learning, AAAI 2015]

Ape-X / R2D2
[Distributed Prioritized Experience Replay, Horgan et al, ICLR 2018]
[Recurrent Experience Replay in Distributed Reinforcement Learning, Kapturowski et al, ICLR 2019]

OpenAI Rapid
**ELF:** RL Framework for Game Research

```python
while True:
    batched_states = GameContext.Wait()
    replies = model(batched_states)
    GameContext.Steps(replies)
```

AlphaGoZero / AlphaZero

- Generate Training data
- Update Models
- Self-Replays

Without human knowledge

$\theta_i$

[Silver et al, Mastering the game of Go without human knowledge, Nature 2017]
Generate Self-play Games

Monte Carlo Tree Search with current model $\theta_i$

Training samples for $\theta_{i+1}$
Update Models

Input features (19x19x17): \((X, Y, X_{-1}, Y_{-1}, \ldots, X_{-7}, Y_{-7}, C')\)

Objective:

\[
J(\theta) = (z - V_\theta)^2 - \pi^T \log p_\theta + c\|\theta\|^2
\]
AlphaGo Zero Strength

• 3 days version
  • 4.9M Games, 1600 rollouts/move
  • 20 block ResNet
  • Defeat AlphaGo Lee.

• 40 days version
  • 29M Games, 1600 rollouts/move
  • 40 blocks ResNet.
  • Defeat AlphaGo Master by 89:11
The Mystery of AlphaZero

• Mystery
  • Is the proposed algorithm really universal?
  • Is the bot almighty? Is there any weakness in the trained bot?

• Lack of Ablation Studies
  • What factor is critical for the performance?
  • Is the algorithm robust to random initialization and changes of hyper parameters?
  • Any adversarial samples?

Impressive Results, No code, No model
ELF OpenGo

• System can be trained with 2000 GPUs in 2 weeks (20 block version)
• Superhuman performance against professional players and strong bots.
• Abundant ablation analysis.

We open source the code and the pre-trained model for the Go and ML community

[ELF OpenGo: An Analysis and Open Reimplementation of AlphaZero, Y. Tian et al, ICML 2019]
ELF OpenGo Performance

Vs top professional players

<table>
<thead>
<tr>
<th>Name</th>
<th>ELO (world rank)</th>
<th>Result</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kim Ji-seok</td>
<td>3590 (#3)</td>
<td>5-0</td>
<td></td>
</tr>
<tr>
<td>Shin Jin-seo</td>
<td>3570 (#5)</td>
<td>5-0</td>
<td></td>
</tr>
<tr>
<td>Park Yeonghun</td>
<td>3481 (#23)</td>
<td>5-0</td>
<td></td>
</tr>
<tr>
<td>Choi Cheolhan</td>
<td>3466 (#30)</td>
<td>5-0</td>
<td></td>
</tr>
</tbody>
</table>

Single GPU, 80k rollouts, 50 seconds
Offer unlimited thinking time for the players

Vs professional players

Single GPU, 2k rollouts, 27-0 against Taiwanese pros.

Vs strong bot (LeelaZero)

[158603eb, 192x15, Apr. 25, 2018]: 980 wins, 18 losses (98.2%)
Distributed ELF (version 1, AlphaGoZero)

Training procedure (8 GPUs)

Selfplay 1 → Selfplay data → Current best model
Selfplay 2 → Selfplay data → Current best model
... → Selfplay data → Current best model

Current best model → Evaluation Server
Update best model and next candidate
Win rate > 55%

Model Zoo

Evaluation 1
Evaluation 2
... → Evaluation m
Distributed ELF (version 1)

Selfplay 1 → Open a port
Selfplay 2 → Receive selfplay data via ZeroMQ
Selfplay n → Current trained model

Training procedure → 8 GPUs

Model Zoo → Evaluation Server
Evaluation 1 → Evaluation 2 → Evaluation m
Distributed ELF (version 1)

Training procedure (8 GPUs)

Selfplay 1 → Selfplay 2 → ... → Selfplay n

Pick the best model and keep selfplaying

Current best model

Model Zoo

300-2k GPUs

Each selfplay client batches 32 parallel games in a batch size of 128

Evaluation 1 → Evaluation 2 → ... → Evaluation m

facebook Artificial Intelligence
Distributed ELF (version 1)

Selfplay 1 → Selfplay 2 → ... → Selfplay n → Training procedure (8 GPUs) → Model Zoo

Evaluation 1 → Evaluation 2 → ... → Evaluation m

Update best model and next candidate
Win rate > 55%

No GPU needed
Distributed ELF (version 1)

Selfplay 1 → Selfplay 2 → ... → Selfplay n → Evaluation 1 ≫ Evaluation 2 ≫ Evaluation m

100 GPUs

Send the current model pairs to evaluate

Training procedure (8 GPUs) → Model Zoo → Each evaluation client batches 2 parallel games

facebook Artificial Intelligence
Distributed ELF (v2)

Putting AlphaGoZero and AlphaZero into the same framework

- AlphaGoZero (more synchronization)
- AlphaZero (less synchronization)

Server controls synchronization
Server also does training.
Next Step: RL Assembly

- Backbone infrastructure for ongoing projects (Hanabi, Bridge, etc)
- Reimplementation of SoTA off-policy RL methods like Ape-X and R2D2
- Incorporate OpenGo and SoTA implementation of MCTS.
- Efficient on single machine (SoTA training FPS so far)

Open source soon
Frame Per Second (FPS) on Atari Games

ReLA: 12.5 KFPS
  using 40 CPU cores + 2 GPU (P100) on a single machine

Ape-X: 12.5 KFPS
  using 360 CPU cores + 1 GPU (distributed system)

• ReLA is GPU bound. Performance is better with more GPUs

• A few more improvements to achieve better performance when releasing.
Architecture

ThreadLoop

Env
Env
Env
Env
Env

Actor

Batch Obs
Batch Action

(Prioritized) Replay Buffer

Data

Mini-Batches & Priority

Update Actor Model

Trainer

Facebook Artificial Intelligence
User Interface (API)

```python
1 env = rela.VectorEnv()
2 for _ in range(num_env_per_thread):
3     game = create_atari(...) 
4     env.append(game)
5 actor = rela.DQNActor(...) 
6 thread = rela.ThreadLoop(actor, env)
```

All objects (env, agent, replay buffer, etc) are created & configured in Python
Model is written in **Python** with **PyTorch’s TorchScript**, and executed in **C++** with multi-threading for maximum throughput.
Native integration with PyTorch C++ API

• Simple/Intuitive manipulations of PyTorch tensors in C++
  • Same as/Similar to Python Interface
  • No extra library needed for operations like downsample/upsampling.

```cpp
1 torch::Tensor s = getObservation();
2 s = s.view({1, 3, height, width});
3 // rescale the image
4 s = torch::upsample_bilinear2d(s, {sHeight, sWidth}, true);
5 s = s.view({3, sHeight, sWidth});
6 // convert to grey scale
7 s = 0.21 * s[0] + 0.72 * s[1] + 0.07 * s[2];
```
Native integration with PyTorch C++ API

• Easier communication between threads/processes via Tensor.
• No extra copy when sending data from/to environments.
Native integration with PyTorch C++ API

• Simultaneous network forwarding at different threads
  • Python GIL becomes irrelevant.
  • No need to block the environment
    • good for simple environments like Go, Bridge, Hanabi and others.
Learning Hand-tuned Heuristics with RL/ML
Combinatorial optimization

Travel Salesman Problem

Job Scheduling

Vehicle Routing

Bin Packing

Protein Folding

Model-Search
Wait...What?

- These are NP-hard problems.
  - No good algorithm unless $P = NP$

- These guarantees are worst-case ones.
  - To prove a lower-bound, construct an adversarial example to fail the algorithm

- For specific distribution, there might be better heuristics.
  - Human heuristics are good but may not be suitable for everything
Direct predicting combinatorial solutions

Convex hull

Seq2seq model

[O. Vinyals et al, Pointer Networks, NIPS 2015]

Policy gradient

Schedule the job to $i$-th slot

Local Rewriting Framework

A learned “gradient descent” that
starts from a feasible solution
iteratively converges to a good solution

How to learn it?

Code: https://github.com/facebookresearch/neural-rewriter
Local Rewriting Framework

Current State (i.e. Solution)

$s_t \xrightarrow{} s_t[\omega_t] \xrightarrow{} s_{t+1}$

Region-Picker

$\omega_t \sim \pi_\omega(\cdot | s_t)$

Rule-Picker

$u_t \sim \pi_u(\cdot | s_t[\omega_t])$

$s_{t+1} = f(s_t, \omega_t, u_t)$
Q-Actor-Critic Training

How to train two policies $\pi_\omega(\cdot \mid s_t)$ and $\pi_u(\cdot \mid s_t [\omega_t])$?

Learn $Q$ to fit cumulative rewards:

$$L_\omega(\theta) = \frac{1}{T} \sum_{t=0}^{T-1} \left( \sum_{t'=t}^{T-1} \gamma^{t'-t} r(s'_{t'}, (\omega'_{t'}, u'_{t'})) - Q(s_t, \omega_t; \theta) \right)^2$$

$\pi_\omega(\cdot \mid s_t)$: Q-learning with soft policy:

$$\pi_\omega(\omega_t | s_t; \theta) = \frac{\exp(Q(s_t, \omega_t; \theta))}{\sum_{\omega_t} \exp(Q(s_t, \omega_t; \theta))}$$

$\pi_u(\cdot \mid s_t [\omega_t])$: Actor-Critic with learned $Q$:

$$L_u(\phi) = -\sum_{t=0}^{T-1} \Delta(s_t, (\omega_t, u_t)) \log \pi_u(u_t | s_t [\omega_t]; \phi)$$

Advantage:

$$\Delta(s_t, (\omega_t, u_t)) = \sum_{t=t'}^{T-1} \gamma^{t'-t} r(s'_{t'}, (\omega'_{t'}, u'_{t'})) - Q(s_t, \omega_t; \theta)$$
How to encode Structure Data

Child-Sum LSTM

\[ y_1 = f(y_2, y_3, x_1) \]

\( f \) can be very complicated:

\[ \tilde{h}_j = \sum_{k \in C(j)} h_k, \]

\[ i_j = \sigma \left( W^{(i)} x_j + U^{(i)} \tilde{h}_j + b^{(i)} \right), \]

\[ f_{jk} = \sigma \left( W^{(f)} x_j + U^{(f)} h_k + b^{(f)} \right), \]

\[ o_j = \sigma \left( W^{(o)} x_j + U^{(o)} \tilde{h}_j + b^{(o)} \right), \]

\[ u_j = \tanh \left( W^{(u)} x_j + U^{(u)} \tilde{h}_j + b^{(u)} \right), \]

\[ c_j = i_j \odot u_j + \sum_{k \in C(j)} f_{jk} \odot c_k, \]

\[ h_j = o_j \odot \tanh(c_j), \]

[Improved Semantic Representation From Tree-Structured Long Short-Term Memory Networks. K. Tai et al]
Applications

(a) $s_t$ 

Online Job Scheduling

(b) $s_t$

Expression Simplification

(c) $s_t$

Vehicle Routing

$\omega_t^* = \arg\max \pi_{\omega}(\cdot, s_t)$

Expression Simplification

Vehicle Routing
Online Job Scheduling

Scheduling 1 (Sequential)

Jobs

Job 1: $T = 2, A = 1$
Job 2: $T = 3, A = 2$
Job 3: $T = 1, A = 3$

Scheduling 2

Graph representation

Resource 1
Resource 2

$T = 2, A = 1$
$T = 3, A = 2$
$T = 1, A = 3$

Graph representation
Online Job Scheduling

Baselines:
- Earliest Job First (EJF)
- Shortest Job First (SJF)
- Shortest First Search (SJFS)
- DeepRM

Offline baselines:
- Google OR-tools (OR-tools)
- SJF-offline

$D$: Number of resources
Online Job Scheduling: Ablation Study

The learned model can generalize to different job distributions.

![Graphs showing average slowdown for different job frequencies, resource distributions, and job lengths.](image-url)
Expression Simplification

Min/Max
Distribution

Min/Max Expansion

3 + 3 → 6
5 ≤ 6 → 1
Expression Simplification

Baselines:
- Z3-simplify
- Z3-ctx-solver-simplify
- Heuristic Search
- Halide rules
Expression Simplification

Transfer learning still works well. A model trained with expression length $\leq 50$ has good performance on test set with expression length $\geq 100$, and better than Z3.
Capacitated Vehicle Routing

![Graph showing average tour length for different vehicle routing problems.](image)

- **Random Sweep**
- **Random CW**
- **Or-tools**
- **Nazari et al. (RL beam 10)**
- **AM (sampling)**
- **NeuRewriter**

<table>
<thead>
<tr>
<th>Problem Set</th>
<th>Average Tour Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>VRP20, Cap30</td>
<td>12.96</td>
</tr>
<tr>
<td>VRP50, Cap40</td>
<td>11.31</td>
</tr>
<tr>
<td>VRP100, Cap50</td>
<td>10.15</td>
</tr>
</tbody>
</table>

**Notes:**
- **VRP** stands for Vehicle Routing Problem.
- **Cap** refers to the capacity constraint.
Coda: An End-to-End Neural Program Decomplier

Cheng Fu¹, Huili Chen¹, Haolan Liu¹, Xinyun Chen³, Yuandong Tian², Farinaz Koushanfar¹, Jishen Zhao¹

¹UC San Diego, ²Facebook AI Research, ³UC Berkeley

NeurIPS 2019
Background: Decompilation

• Goal of Decompilation
  • From Binary Execution to High-level program language
Challenges

• Many hardware architectures (ISA): x86, MIPS, ARM

• Many Programming Languages (PL)
  • Extra Human effort to extend to the new version of the hardware architectures or programming languages

• Our goals:
  • Maintain both the functionality and semantics of the binary executables
  • Make the design process end-to-end (generalizable to various ISAs and PLs)
Coda Design

Leverage both syntax and dynamic information

Find good candidates

Iteratively correct the candidates towards perfect match

Low-level code

End-to-End Framework

High level program

Stage 1

Stage 2

Code Sketch Generation

Error Correction

facebook Artificial Intelligence
Stage 1: Coda Sketch Generation

- Is Decompilation simply a translation problem?

More than a translation problem!
Stage 1: Coda Sketch Generation

- **Encoder**
  - N-ary Tree Encoder to capture inter and intra dependencies of the low-level code.
  - Opcode and its operands are encoded together
  - Different encoder is used for different instruction types
    - memory (mem)
    - branch (br)
    - arithmetic (art).

```
# source C program
a = b * c;
if(a > c){
c = a * c - b;
}
```

<table>
<thead>
<tr>
<th>Mem Address</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>lw $1, 24($fp)</td>
</tr>
<tr>
<td>1</td>
<td>lw $2, 20($fp)</td>
</tr>
<tr>
<td>2</td>
<td>mul $1, $1, $2</td>
</tr>
<tr>
<td>3</td>
<td>sw $1, 28($fp)</td>
</tr>
<tr>
<td>4</td>
<td>lw $1, 28($fp)</td>
</tr>
<tr>
<td>5</td>
<td>lw $2, 20($fp)</td>
</tr>
<tr>
<td>6</td>
<td>slt $1, $2, $1</td>
</tr>
<tr>
<td>7</td>
<td>beqz $1, $BB0_3</td>
</tr>
<tr>
<td>8</td>
<td>j $B2</td>
</tr>
<tr>
<td>9</td>
<td>$B2:</td>
</tr>
<tr>
<td>10</td>
<td>lw $1, 28($fp)</td>
</tr>
<tr>
<td>11</td>
<td>lw $2, 20($fp)</td>
</tr>
<tr>
<td>12</td>
<td>mul $1, $1, $2</td>
</tr>
<tr>
<td>13</td>
<td>lw $2, 24($fp)</td>
</tr>
<tr>
<td>14</td>
<td>subu $1, $1, $2</td>
</tr>
<tr>
<td>15</td>
<td>j $B3</td>
</tr>
<tr>
<td>16</td>
<td>sw $1, 20($fp)</td>
</tr>
</tbody>
</table>
Stage 1: Coda Sketch Generation

- **Decoder**
  - Generate Abstract Syntax Tree (AST)
  - AST can be equivalently translated into its corresponding high level Program
  - Advantages:
    - Prevent error propagation/Preserve node dependency/capture PL grammar
    - Boundaries are more explicit (terminal nodes)
  - Using Attention Mechanism
Stage 2: Iterative Error Correction

- The sketch generated in Stage 1 may contain errors:
  - mispredicted tokens, missing lines, redundant lines

<table>
<thead>
<tr>
<th>Golden program</th>
<th>Wrongly predicted</th>
<th>Missing lines</th>
<th>Redundant lines</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>If( a &gt; c ) {</code></td>
<td><code>If( a &gt; b ) {</code></td>
<td><code>If( a &gt; c ) {</code></td>
<td><code>If( a &gt; c ) {</code></td>
</tr>
<tr>
<td><code>a = b + c * a;</code></td>
<td><code>a = b + c * a;</code></td>
<td><code>a = b + c * a;</code></td>
<td><code>a = b + c * a;</code></td>
</tr>
<tr>
<td><code>b = a - c;</code></td>
<td><code>b = a - b;</code></td>
<td><code>b = a;</code></td>
<td><code>b = a;</code></td>
</tr>
<tr>
<td><code>}</code></td>
<td><code>}</code></td>
<td><code>}</code></td>
<td><code>}</code></td>
</tr>
</tbody>
</table>
Stage 2: Iterative Error Correction

• Correct the error by iterative Error Predictor (EP)
  • Iterative rewriting!
  • Spot errors in the generated assembly codes
  • Fix errors and recompile
  • Repeat 10 times
Experimental Setup

- Compiler configuration: Clang `-O0` (no code optimization)
- Benchmarks:
  - Synthetic programs:
    - **Karel library (Karel)** – only function calls
    - **Math library (Math)** – function calls with arguments
    - **Normal expressions (NE)** – `(\^,\&,\*,\-,\!,\<\>,\|,\% ....)`
    - **Math library + Normal expressions (Math+NE)** – replaces the variables in NE with a return value of math function.
- Metrics:
  - Token Accuracy
  - Program Accuracy
Result – Stage 1 Performance

• Token accuracy (%) across benchmarks

<table>
<thead>
<tr>
<th>Benchmarks</th>
<th>Seq2Seq</th>
<th>Seq2Seq+Attn</th>
<th>Seq2AST+Attn</th>
<th>Inst2seq+Attn</th>
<th>Inst2AST+Attn</th>
</tr>
</thead>
<tbody>
<tr>
<td>Karel&lt;sub&gt;S&lt;/sub&gt;</td>
<td>51.61</td>
<td>97.13</td>
<td>99.81</td>
<td>98.83</td>
<td>99.89</td>
</tr>
<tr>
<td>Math&lt;sub&gt;S&lt;/sub&gt;</td>
<td>23.12</td>
<td>94.85</td>
<td>99.12</td>
<td>96.20</td>
<td>99.72</td>
</tr>
<tr>
<td>NE&lt;sub&gt;S&lt;/sub&gt;</td>
<td>18.72</td>
<td>87.36</td>
<td>90.45</td>
<td>88.48</td>
<td>94.66</td>
</tr>
<tr>
<td>(Math+NE)&lt;sub&gt;S&lt;/sub&gt;</td>
<td>14.14</td>
<td>87.86</td>
<td>91.98</td>
<td>89.67</td>
<td>97.90</td>
</tr>
<tr>
<td>Karel&lt;sub&gt;L&lt;/sub&gt;</td>
<td>33.54</td>
<td>94.42</td>
<td>98.02</td>
<td>98.12</td>
<td>98.56</td>
</tr>
<tr>
<td>Math&lt;sub&gt;L&lt;/sub&gt;</td>
<td>11.32</td>
<td>91.94</td>
<td>96.63</td>
<td>93.16</td>
<td>98.63</td>
</tr>
<tr>
<td>NE&lt;sub&gt;L&lt;/sub&gt;</td>
<td>11.02</td>
<td>81.80</td>
<td>85.92</td>
<td>85.97</td>
<td>91.92</td>
</tr>
<tr>
<td>(Math+NE)&lt;sub&gt;L&lt;/sub&gt;</td>
<td>6.09</td>
<td>81.56</td>
<td>85.32</td>
<td>86.16</td>
<td>93.20</td>
</tr>
</tbody>
</table>

• Highest token accuracy across all benchmarks (96.8% on average) compared to baselines.
• 10.1% and 80.9% margin over a naive Seq2Seq model with and without attention.
• More tolerant to the growth of program length.
## Result – Stage 2 Performance

- **Program accuracy (%)**

<table>
<thead>
<tr>
<th>BenchMarks</th>
<th>(i) Error Detection</th>
<th>(ii) Before EC</th>
<th>After EC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>s2s,10</td>
<td>i2a,10</td>
<td>s2s</td>
</tr>
<tr>
<td></td>
<td>s2s,10</td>
<td>i2a,10</td>
<td>i2a</td>
</tr>
<tr>
<td></td>
<td>s2s</td>
<td>i2a</td>
<td></td>
</tr>
<tr>
<td>Math(_S)</td>
<td>91.4</td>
<td>94.2</td>
<td>40.1</td>
</tr>
<tr>
<td>NE(_S)</td>
<td>83.5</td>
<td>88.7</td>
<td>6.6</td>
</tr>
<tr>
<td>(Math+NE)(_S)</td>
<td>83.6</td>
<td>90.1</td>
<td>3.5</td>
</tr>
<tr>
<td>Math(_L)</td>
<td>87.5</td>
<td>91.3</td>
<td>21.7</td>
</tr>
<tr>
<td>NE(_L)</td>
<td>78.1</td>
<td>84.5</td>
<td>0.2</td>
</tr>
<tr>
<td>(Math+NE)(_L)</td>
<td>80.2</td>
<td>85.3</td>
<td>0.1</td>
</tr>
</tbody>
</table>

- **s2s** = sequence-to-sequence with attention
- **I2a** = instruction encoder to AST decoder with attention

Baseline

Ours
Summarization and Future Works

• Summary
  • Gives examples of scalable RL system
  • RL/ML can be used to learn heuristics for system

• Large Open Space Ahead
  • ML captures statistics regularity and leads to better solutions
  • Application to large-scale systems?
  • Theoretical Guarantees?
Thanks!