Lecture 14:
Simulating Virtual Worlds to Train Agents

Visual Computing Systems
Stanford CS348K, Spring 2023
Today

- We’ve talked about how ML/AI techniques are used to improve visual computing applications: computational photography, rendering, video compression, etc…

- Today we’ll talk about how rendering and simulation techniques are increasingly central to the progress of AI
Think back to earlier in course

What was the biggest practical bottleneck to training good models?

Snorkel: Rapid Training Data Creation with Weak Supervision

Alexander Ratner  Stephen H. Bach  Henry Ehrenberg
Jason Fries  Sen Wu  Christopher Ré
Stanford University  Stanford, CA, USA
{ajratner, bach, henryre, jfries, senwu, chrismre}@cs.stanford.edu

ABSTRACT

Labeling training data is increasingly the largest bottleneck in deploying machine learning systems. We present Snorkel, a first-of-its-kind system that enables users to train state-of-the-art models without hand labeling any training data. Instead, users write labeling functions that express arbitrary heuristics, which can have unknown accuracies and correlations. Snorkel denoises their outputs without access to ground truth by incorporating the first end-to-end implementation of our recently proposed machine learning paradigm, data programming. We present a flexible interface layer for writing labeling functions based on our experience over the past year collaborating with companies, agencies, and research labs. In a user study, subject matter experts build models 2.8× faster and increase predictive performance an average 45.5% versus seven hours of hand labeling. We study the modeling tradeoffs in this new setting and propose an optimizer for automating tradeoff decisions that gains up to 1.8× speedup per pipeline execution. In two collaborations, with the U.S. Department of Veterans Affairs and the U.S. Food and Drug Administration, and on four open-source text and image data sets representative of other deployments, Snorkel provides 132% average.

Figure 1: In Example 1.1, training data is labeled by sources of differing accuracy and coverage. Two key challenges arise in using this weak supervision effectively. First, we need a way to estimate the unknown source accuracies to resolve disagreements. Second, we need to pass on this critical lineage information to the end model being trained.
Using rendering/simulation to generate supervision to train better models

Part 1:
Use high-fidelity simulation to simulate what sensors would measure in hypothetical real-world situations (Improving perception systems)
Stanford Gibson project: acquire/render real world data

- Dataset acquired via 3D scanning (3D mesh + texture)
- Geometry, normals, semantics, + (so-called) “photorealistic” 3D
Enhancing CG images to look like real-world images using image-to-image transfer
Modifying real-world images to create novel situations

Remove or move this car.
Part 2:
Training agent behaviors using reinforcement learning
RL in 30 seconds

Model Inference

environment
observation
e.g. RGB image

\[ \pi \theta \]

agent
action
**RL in 30 seconds**

**Model Inference**
- Environment observation (e.g., RGB image) → \( \pi \theta \) → Agent action

**Model Training**
- Sequence of observations
- Sequence of agent actions
- Compute loss gradients → \( \pi \theta \) → Update model via SGD
- Reward: change in distance from goal
RL in 30 seconds

Model Inference

environment observation e.g. RGB image

\[ \pi \theta \rightarrow \text{agent action} \]

Model Training

Rollout

compute loss gradients

\[ \pi \theta \rightarrow \text{update model via SGD} \]
RL in 30 seconds

Many rollouts:
- Agents independently navigating same environments

Batch Model Training

Rollout 0
Rollout 1
Rollout 2
...  
Rollout N-1

compute loss gradients

$\pi \theta$

update model via SGD
**RL in 30 seconds**

Many rollouts:
- Agents independently navigating the same environments
- Or different environments

---

**Batch Model Training**

Rollout 0
Rollout 1
Rollout 2
Rollout 3
Rollout 4
Rollout 5
...  
Rollout N-1

compute loss gradients

update model via SGD
Learning robot skills requires many trials (billions) of learning experience

- Training in diverse set of virtual environments
- Many training trials in each environment
Need significant amounts of simulated experience to learn skills

Example: even for simple PointGoal navigation task: need billions of steps of “experience” to exceed traditional non-learned approaches
Physics simulation
Random incoming ball + Random target

Practice in simulation...
Many interactive virtual home environments

- Navigate to a location
- Find an object
- Rearrange the room so objects are in desired locations
- Pour oneself a glass of milk
Multi-agent games

Hanabi (Card Game)

Overcooked (Sims-Like Env)

Atari Games

Basketball Pong

Boxing

Combat Plane

Combat Tank

Double Dunk

Entombed Competitive

Entombed Cooperative

Flag Capture

Foozpong

Ice Hockey

Joust

Mario Bros
RL workload summary

- **Within a rollout**
  - For each step of a rollout:
  - Render $\rightarrow$ Execute policy inference $\rightarrow$ simulate next world state

- **Across *many* independent rollouts**
  - Simulated agents may (or may not) share scene state
  - Diversity in scenes in a batch of rollouts is desirable to avoid overfitting, sample efficiency of learning
Common simulation approach: treat simulator as a black box, gain high throughput via scale-out parallelization.

Treat existing simulation engines as a black box.

Run many copies of the black box in parallel.
OpenAI's “OpenAI 5” Dota 2 bot

<table>
<thead>
<tr>
<th>Description</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPUs</td>
<td>128,000 preemptible CPU cores on GCP</td>
</tr>
<tr>
<td>GPUs</td>
<td>256 P100 GPUs on GCP</td>
</tr>
<tr>
<td>Experience collected</td>
<td>~180 years per day (~90 years per day counting each hero separately)</td>
</tr>
<tr>
<td>Size of observation</td>
<td>~36.8 kB</td>
</tr>
<tr>
<td>Observations per second of gameplay</td>
<td>7.5</td>
</tr>
<tr>
<td>Batch size</td>
<td>1,048,576 observations</td>
</tr>
<tr>
<td>Batches per minute</td>
<td>~60</td>
</tr>
</tbody>
</table>
Generating simulated experience is computationally demanding

**OpenAI Five**

Rapid: 128,000 CPUs, days of training

**Navigation in 3D scanned environments**

64 GPUs over 2.5 days (2B experience samples)

**Game playing**

Deepmind Lab Training with 4000 CPUs and 64 TPUs
Example: PointGoal navigation task system components

- **World State “Simulator”**: (updates position of agent in scene, detects collisions with scene geometry)
- **Renderer**: (render scene from viewpoint of agent)
- **Inference/Learning**: (inference: action from rendered image, learning: update policy model from rollouts) $\pi_\theta$

Diagram:
- Database of 3D assets (meshes, textures, collision meshes)
- Viewpoints, scene object positions
- Non-rendered state: position, compass…
- Rendered frames
- Next action
Basic design: parallelize over workers

Ask yourself:
1. What data gets communicated?
2. Can the system scale to sufficient parallelism?
3. Are there sync bottlenecks

Learning
learning: update policy model from rollouts

\( \pi \theta \)
Example: Rapid (OpenAI)

**Optimizer + Connected Rollout Workers (x256)**

**Rollout Workers**
- ~500 CPUs
- Run episodes
  - 80% against current bot
  - 20% against mixture of past versions
- Randomized game settings
- Push data every 60s of gameplay
  - Discount rewards across the 60s using generalized advantage estimation

**Optimizer**
- 1 p100 GPU
- Compute Gradients
  - Proximal Policy Optimization with Adam
  - Batches of 4096 observations
  - BPTT over 16 observations

**Eval Workers**
- ~2500 CPUs
- Play in various environments for evaluation
  - vs hardcoded "scripted" bot
  - vs previous similar bots (used to compute Trueskill)
  - vs self (for humans to watch and analyze)
What modern graphics engines are designed to render

- 4K image outputs
- 30-60 fps
- Advanced lighting and material simulation
Low-resolution images with pre-captured lighting (from Gibson): clearly not state-of-the-art rendering! ;-)}
Inefficient simulation/rendering: rendering a small image does not make good use of a modern GPU (rendering throughput is low)

Duplication of computation and memory footprint (for scene data) across renderer/simulator instances
Often the best way to reduce communication / increase efficiency is to make the most efficient use out of one node.

Can we make simulation faster?
Trend: cracking open the black box: 100x end-to-end training speedups using optimized “batch simulators”

New simulators specifically designed to efficiently execute 1000’s of worlds at once on a single GPU or multi-core CPU

**BPS3D: Pointgoal Navigation**

~1000 Worlds per GPU

**CuLE: Atari on the GPU**

~5000 Worlds per GPU

**Isaac Gym: GPU Physics**

~10000 Worlds per GPU
Main idea: design a renderer that executes rendering for 100s-1000’s of unique rollouts in a single request

Inference/training, simulation, and rendering all operate on batches of N requests (rollouts)

Efficient bulk communication between three components
Opportunities provided by a batch rendering interface

- Wide parallelism: rendering each scene in a batch is independent
  - "Fill up" large parallel GPU with rendering work
  - Enables graphics optimizations like pipelining frustum culling (removing off-screen geometry before drawing it) for one environment with rendering of another

- Footprint optimizations: rendering requests in a batch can share same geometry assets
  - Significantly reduces memory footprint, enables large batch size
  - $N \sim 256$-1024 (per GPU) in our experiments: fills up large GPU
  - Limit number of unique scenes in a batch to $K \ll N$ scenes.
    - GPU RAM and scene size determines $K$

- Amortize communication: rendering requests in a batch can be packaged and drawn together
  - Render frames in batch to tiles in a single large frame buffer to avoid state update
2.5B frames of experience in 48 hrs on a single RTX 3090

“PointGoal Navigation” in Gibson environments (Habitat Labs)

Run render->infer->train loop runs at 13,000 fps per GPU

"Large Batch Simulation for Deep Reinforcement Learning", B. Shacklett, E. Wijmans, A. Petrenko, M. Savva, D. Batra, V. Koltun, K. Fatahalian, ICLR 2021
**NVIDIA Issac Gym**

- Same idea of batched many-environment execution, applied to physics
- Simulate 100's to 1000's of world environments simultaneously on the GPU
- Current state for all environments packaged in a single PyTorch tensor
- User can write GPU-accelerated loss/reward functions in PyTorch on this tensor
- Result: tight loop of simulate/infer/train
Tonight’s reading

- How would you design game engine APIs to support creation of custom games that execute efficiently in “batch simulation” mode
- All game logic, not just rendering and physics
Part 3:
Growing interest in general purpose agent skills
Interesting end-to-end systems research questions

- How much fidelity is needed to train models that successfully transfer into the real-world?
  - Do we even need photorealistic quality (or advanced physics) to train policies that work in the real world?

- Is it better to spend compute on higher fidelity simulation, or use the budget to train on a wider range of simulations, or even many tasks?
Example Sim2Real experiments: RoboTHOR

Virtual environment

Real world photo of corresponding environment (in lab)

[Dietke 20]
Understanding the effects of sim2real gap
What parts of real-world sensing do we really need to model in simulation?

Example visual corruptions

- Clean
- Defocus Blur
- Motion Blur
- Spatter
Value of diversity of scenes

Example: ProcTHOR

Procedurally generated floorpans, furniture arrangements, random material assignments, etc.
Greater diversity of scenes wins

Better off training on a large number of highly diverse scenes than a small number of photorealistic ones
Value of diversity of tasks

DeepMind’s XLand: procedurally generate terrains and the rules of the game.

Wide range of “games” requiring different strategies and skills can be defined by specifying goals for agents

Example 1: agent 1 must find a yellow sphere and hold it, while agent 2 must stand by a yellow pyramid

\[ g_1 := \text{hold}(\text{me}, \text{yellow sphere}) \]
\[ g_2 := \text{near}(\text{me}, \text{yellow pyramid}) \]

Example 2: goals for hide and seek: Agent 1 should move to see its opponent. Agent 2’s goal is to not be seen by agent 1.

\[ g_1 := \text{see}(\text{me}, \text{opponent}) \]
\[ g_2 := \text{not}(\text{see}(\text{opponent}, \text{me})) \]

Example 3: encourage teamwork. Give both agents the same goal of moving one object by another

\[ g_1 := \text{near}(\text{yellow pyramid}, \text{yellow sphere}) \]
\[ g_2 := \text{near}(\text{yellow pyramid}, \text{yellow sphere}) \]

The result of this training process is an agent that is generally capable across the held-out evaluation space. Qualitatively, we observe the agent exhibiting behaviours that are generally applicable, rather than optimal for any specific task. Examples of such behaviours include: experimentation through directed exploration until the agent recognises a rewarding state has been achieved; seeking another player out to gather information of its state irrespective of its goal; and tagging another player if it is holding an object that is related to the agent’s goal irrespective of that player’s intention. We also probe quantitatively the behaviour of agents in test-time multi-agent situations and see evidence of cooperation emerging with training. In addition to the agent exhibiting zero-shot capabilities across a wide evaluation space, we show that finetuning on a new task for just 100 million steps (around 30 minutes of compute in our setup) can lead to drastic increases in performance relative to zero-shot, and relative to training from scratch which often fails completely.
Value of “open worlds”: many possible tasks

Example: learning to perform many MineCraft tasks (MineDojo)
Takeaways

- Higher performance simulation = higher fidelity simulation
  - Interest in high-fidelity simulation to learn skills from inputs that similar to real world inputs (minimize “sim2real” gap)
  - The hope is that skills transfer to real world

- Higher performance simulation = many more [lower fidelity] simulations
  - To learn “foundational” skills and knowledge, evidence suggests more diversity in environments and tasks is better
  - Train pretty-good-across-the-board “generalist” agents that are a good starting point to be quickly “fine tuned” or adapted to specific tasks in the future