Lecture 12:

Creating AI Agents + Simulating Virtual Worlds for Training

Visual Computing Systems
Stanford CS348K, Spring 2024
First:
resuming discussion of LLM-based agents from last time
(Generative agents + Voyager papers)
LLM-driven problem solving vs. trial and error

- Problem solving approach of the previous lecture and our discussion up until now:
  - State problem in plain text
  - Use LLM as a general purpose problem solver
  - LLM provides text-based (or code based) solution strategy
  - Execute strategy in a virtual world

- Now let’s consider problems where it’s less obvious how to describe the problem in text...
Dexterous manipulation
Playing an FPS game...
Random incoming ball + Random target
Why learning via trial and error requires a lot of simulated experience (reinforcement learning example)
RL in 30 seconds

Model Inference

environment observation e.g. RGB image $\rightarrow \pi \theta \rightarrow$ 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
- Update model via SGD

Reward: change in distance from goal
RL in 30 seconds

Model Inference

- Environment
- Observation (e.g. RGB image)
- \( \pi \theta \) (policy)
- Agent action

Model Training

- Rollout
- Compute loss gradients
- 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
Many rollouts:
- Agents independently navigating same environments
- Or different environments

Batch Model Training

Rollout 0 ➔ compute loss gradients ➔ update model via SGD ➔ Rollout 1 ➔ Rollout 2 ➔ Rollout 3 ➔ Rollout 4 ➔ Rollout 5 ➔ Rollout N-1
Learning skills can require 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.
Accurate Physics simulation
Random incoming ball + Random target

Practice in simulation...
Entity Component System (ECS):

- Design pattern from game industry - improve performance & maintainability of game logic
- Current implementations target standard game engine workload: 1 world at ~60 fps

Video Credit: OpenAI
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

Fozzpong

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>OPENAI FIVE</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPUs</td>
</tr>
<tr>
<td>GPUs</td>
</tr>
<tr>
<td>Experience collected</td>
</tr>
<tr>
<td>Size of observation</td>
</tr>
<tr>
<td>Observations per second of gameplay</td>
</tr>
<tr>
<td>Batch size</td>
</tr>
<tr>
<td>Batches per minute</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
Large-scale agent training is expensive!

OpenAI Five
Robotics in Virtual World
OpenAI Hide and Seek

Learning Dota 2: Months of training
64 GPUs over 2.5 days (2B experience samples)
High-level strategies emerge after billions of world time steps

<table>
<thead>
<tr>
<th>CPUs</th>
<th>128,000 preemptible CPU cores on GCP</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPUs</td>
<td>256 P100 GPUs on GCP</td>
</tr>
<tr>
<td>Experience collected</td>
<td>~180 years per day (~900 years per day counting each hero separately)</td>
</tr>
</tbody>
</table>
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)

- **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)

πθ

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

Model Parameters

Optimizers use NCCL2 to average gradients at every step.
What modern graphics engines are designed to render

- 4K image outputs
- 30-60 fps
- Advanced lighting and material simulation
Example renderings from common RL learning environments
- Low resolution
- Simple lighting/shading
Design issues

- 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

- Seems wasteful, right?
Tonight’s reading

- The design of a game engine for the specific case of running many independent world simulations at the same time on a GPU [Shacklett et al 2023]