Soft Actor-Critic Agent in MineRL

Jacob Santoni, Liam Orr, & Rohith Reddy
CSE 410: Reinforcement Learning
University at Buffalo
December 8th, 2020

https://github.com/ContemporaryArtwork/MineRL_NavigateDenseDQNAgent

©2020, All Rights Reserved
Topics for Discussion

❖ Project Description
❖ Background
❖ Implementation
❖ Demo
❖ Results
❖ Key Observations / Summary
❖ Thank you Page
Project Description

- We attempted to solve **MineRLNavigateDense-v0**, an environment from MineRL
- Trying to navigate from a spawn point to another point farther away
- Part of MineRL, a competition to develop **sample efficient** algorithms
Background: Malmö, Minecraft, and MineRL

- **Minecraft**: It’s a game you all probably know well
- **Malmö**: Reinforcement learning backend for Minecraft, by Microsoft
- **MineRL**: A competition framework built on top of Malmö which gives us a set of challenge environments and user generated data
- Overall goal of the MineRL competition is to produce an agent capable of mining diamonds, displaying **sample efficiency**
Background: MineRLNavigateDense-v0

- Environment from MineRL
- Agent has to navigate from a spawn point to another point on the map
- Has a compass which always points at the goal
- 3x64x64 representation of the POV of the agent
- Dense version gives small positive rewards for getting closer to the goal and negative for going farther away
Implementation: Soft Actor-Critic

❖ Spaces

Observation Space = Compass Angle + POV

Action Space = Yaw (Continuous)

❖ Objective - Maximize Expected Return & Maximize Entropy

❖ Exploration vs. Exploitation - Controlled by alpha parameter which scales entropy

❖ Large alpha -> Large entropy -> Large exploration

Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$

Compute targets for the Q functions:

$$y(r, s', d) = r + \gamma (1 - d) \left( \min_{i=1,2} Q_{i,\text{target}}(s', \tilde{a}') - \alpha \log \pi_\theta(\tilde{a}' | s') \right), \quad \tilde{a}' \sim \pi_\theta(\cdot | s')$$

Update Q-functions by one step of gradient descent using

$$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} (Q_{i,\theta}(s, a) - y(r, s', d))^2 \quad \text{for } i = 1, 2$$

Update policy by one step of gradient ascent using

$$\nabla_{\theta} \frac{1}{|B|} \sum_{s \in B} \left( \min_{i=1,2} Q_{i,\phi_i}(s, \tilde{a}_\theta(s)) - \alpha \log \pi_\theta(\tilde{a}_\theta(s) | s) \right),$$

where $\tilde{a}_\theta(s)$ is a sample from $\pi_\theta(\cdot | s)$ which is differentiable wrt $\theta$ via the reparametrization trick.

Update target networks with

$$\phi_{\text{target},i} \leftarrow \rho \phi_{\text{target},i} + (1 - \rho) \phi_i \quad \text{for } i = 1, 2$$
Implementation Details

❖ **Replay Buffer**
   Improve Sample Efficiency

❖ **Double Q-Network**
   Makes Learning More Stable

❖ **Freeze Target Critics**
   Breaking Critic/Target Correlations

❖ **Stochastic Policy**
Demo (Using POV)

Deterministic Testing

Trapped Agent During Evaluation

https://youtu.be/s7ab244Iwp0

https://youtu.be/bBk45epnijn8
Results Contextualized

❖ When training our agent, we found that it would start off likely achieving high reward or even **finishing the episode successfully**, but after many episodes it would average out to good (i.e. positive) but less than ideal reward

❖ Due to the complexity of the environment and the relatively small training, it is hard to assess if this is due to the randomness involved or if the agent is forgetting or ignoring the optimal policy.
### Results Benchmark

<table>
<thead>
<tr>
<th></th>
<th>Treechop</th>
<th>Navigate (S)</th>
<th>Navigate (D)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DQN (Minh et al., 2015)</td>
<td>3.73 ± 0.61</td>
<td>0.00 ± 0.00</td>
<td>55.59 ± 11.38</td>
</tr>
<tr>
<td>A2C (Minh et al. 2016)</td>
<td>2.61 ± 0.50</td>
<td>0.00 ± 0.00</td>
<td>-0.97 ± 3.23</td>
</tr>
<tr>
<td>Behavioral Cloning</td>
<td><strong>43.9 ± 31.46</strong></td>
<td>4.23 ± 4.15</td>
<td>5.57 ± 6.00</td>
</tr>
<tr>
<td>PreDQN</td>
<td>4.16 ± 0.82</td>
<td>6.00 ± <strong>4.65</strong></td>
<td><strong>94.96 ± 13.42</strong></td>
</tr>
<tr>
<td>Human</td>
<td>64.00 ± 0.00</td>
<td>100.00 ± 0.00</td>
<td>164.00 ± 0.00</td>
</tr>
<tr>
<td>Random</td>
<td>3.81 ± 0.57</td>
<td>1.00 ± 1.95</td>
<td>-4.37 ± 5.10</td>
</tr>
</tbody>
</table>

Table 2: Results in Treechop, Navigate (S)parse, and Navigate (D)ense, over the best 100 contiguous episodes. ± denotes standard deviation. Note: humans achieve the maximum score for all environments shown.
Results

<table>
<thead>
<tr>
<th></th>
<th>No POV</th>
<th>POV</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average Reward</td>
<td>34</td>
<td>19</td>
</tr>
<tr>
<td>Number Episodes</td>
<td>25</td>
<td>10</td>
</tr>
<tr>
<td>Best Episode</td>
<td>151</td>
<td>54</td>
</tr>
</tbody>
</table>

Without POV Observation

With POV Observation

Episodic Reward

Reward vs. Episode

Episodic Reward

Reward vs. Episode
Episode Renders

-18.1 Reward
https://youtu.be/DcEFFxwzV44

8.9 Reward
https://youtu.be/jQ5ZU--qJPA

148.4 Reward
https://youtu.be/0K3tUOaLSJ0
Key Observations / Summary

❖ Minecraft has an incredibly large Observation Space
❖ POV: Less sample efficient, but **more robust agent**.
❖ Training = **Resource Intensive**
  With CNN ~10 minutes per episode
  Larger replay buffer -> more stable training
  Malmo rendering incomplete / crashes
❖ Hyper-Parameter Sensitive
❖ Ideas for Improvement
  Distributed Learning
  Use MineRL Dataset
References


Thank You!!!