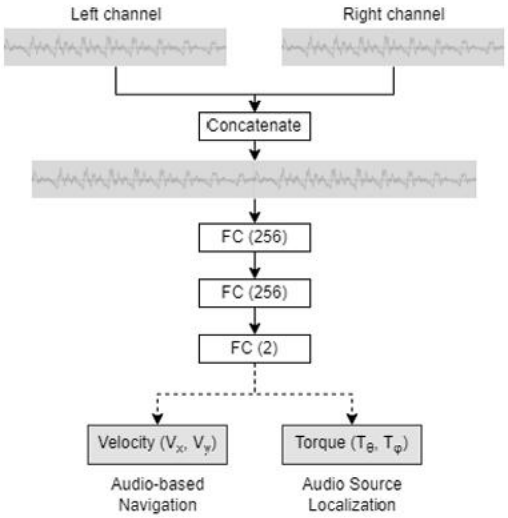
Project Report:
How offline learning can affect an audio navigation environment

Collin Reisman

Introduction

Audio navigation is essential in the implementation of many technologies from the creation of voice to text, to sound filtering. However, the research to locate people by voice has not yet seen a proper real-world implementation. The purpose of this experiment is to evaluate the effectiveness of an off-policy model when navigating in virtual three-dimensional audio-based spaces. Audio based environments are known for their extreme noise, presenting an opportunity for offline model's use of a replay buffer to create a legitimate increase in the model's ability to echolocate.

Method



Model of the feed forward network [1]

The method of this experiment is mostly mirrored from the original experiment [1]. Since the original repo only had the resources for audio navigation and excluded its localization environment. The experiment will be conducted exclusively on its ability to navigate the audio-based environment. The network used for this experiment is a feed forward network which receives both right and left channels of input and concatenates them into a single vector containing 2048 samples. These samples are the info inputted into the models. [1]. The output produced by the models is the vector values of x & y to determine the translation of the agent.

Dataset

The dataset was created using 5 audio books that recorded using a voice activity detector totaling around 500 utterances total per speaker for the audio navigation environment, the utterances are split 400 for training and 100 for the test set.

Agents

For this experiment, the baseline will be the standard PPO model[2]. This is because this model was the original model in the experiment to create a fair baseline when comparing offline vs online. The models will train for around 5-10 million episodes unless training stalls. Other models that will be tested are TD3 and soft actor critic. Both models are off policy models which utilize a replay buffer. Since the TD3 model will be implemented as a custom PyTorch model, there is risk that poor implementation may cause poor training, creating unfair evaluation for offline learning. For this reason, the SAC model will serve as a control group to ensure a fair evaluation of offline vs online learning for more than just TD3. All agents follow a reward function of +1 for reaching the target speaker, -1 for falling out of bounds or going to the wrong speaker, and -.001 per step to encourage exploration.

The soft actor critic model implemented uses unities ml agent's library follows the basic parameters of SAC where it[3]:

samples the policy and transition and stores it in the replay buffer

$$\begin{aligned}
&\mathbf{a}_t \sim \pi_\phi(\mathbf{a}_t|\mathbf{s}_t) \\
&\mathbf{s}_{t+1} \sim p(\mathbf{s}_{t+1}|\mathbf{s}_t, \mathbf{a}_t) \\
&\mathcal{D} \leftarrow \mathcal{D} \cup \{(\mathbf{s}_t, \mathbf{a}_t, r(\mathbf{s}_t, \mathbf{a}_t), \mathbf{s}_{t+1})\}
\end{aligned}$$

Then in each gradient step: it updated the Q parameters, policy weights, and target network weights:

$$\begin{aligned}
&\theta_i \leftarrow \theta_i - \lambda_Q \hat{\nabla}_{\theta_i} J_Q(\theta_i) \text{ for } i \in \{1, 2\} \\
&\phi \leftarrow \phi - \lambda_\pi \hat{\nabla}_\phi J_\pi(\phi) \\
&\alpha \leftarrow \alpha - \lambda \hat{\nabla}_\alpha J(\alpha) \\
&\bar{\theta}_i \leftarrow \tau\theta_i + (1-\tau)\bar{\theta}_i \text{ for } i \in \{1, 2\}
\end{aligned}$$

Creating the TD3 model

One of the main goals of this project was to design and implement my own model into the PyTorch library. This was the TD3 model that was the focal point of this paper. To create this model, I used the previously made architecture of the unity SAC model and swapped components consistent with TD3[4].

I changed the policy to a deterministic policy:

$$\nabla_\phi J(\phi) = N^{-1} \sum \nabla_a Q_{\theta_1}(s,a)|_{a=\pi_\phi(s)} \nabla_\phi \pi_\phi(s)$$

Applied simple Gaussian noise for exploration:

$$: a \sim \pi_\phi(s) + \epsilon,$$

$$\epsilon \sim \mathcal{N}(0, \sigma)$$

Added Clipped double q-learning, target policy smoothing, and Delayed policy updates:

Sample mini-batch of N transitions (s, a, r, s') from $\mathcal{B}$
$\tilde{a} \leftarrow \pi_{\phi'}(s') + \epsilon, \quad \epsilon \sim \text{clip}(\mathcal{N}(0, \tilde{\sigma}), -c, c)$
$y \leftarrow r + \gamma \min_{i=1,2} Q_{\theta_i'}(s', \tilde{a})$
Update critics $\theta_i \leftarrow \text{argmin}_{\theta_i} N^{-1} \sum (y - Q_{\theta_i}(s,a))^2$

Testing Methodology

The tests for this experiment will be conducted as 100 episodes per model in the specific test environment. This environment is a switch in the unity program which switches to a separate set of actor's voices, the target voice will still be the same as the voice during training, however relying on a unique set of samples. For this part of the project, I edited the listening agent file to store the success rate once it has completed 100 episodes to properly measure the success rate of each model. While the model is trained in a headless environment, the test will be conducted in a fully simulated and rendered environment to properly examine the actions of each model as well as the fully rendered environment is a more accurate representation of how the models would react in a real environment.

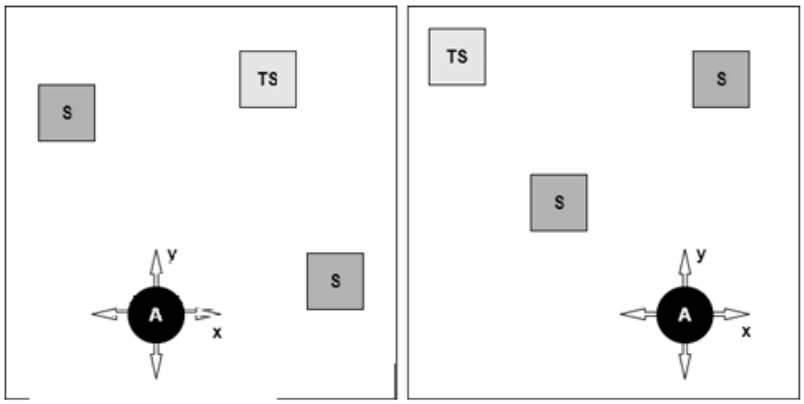


Figure 1 showing design of audio navigation model [1]

Results

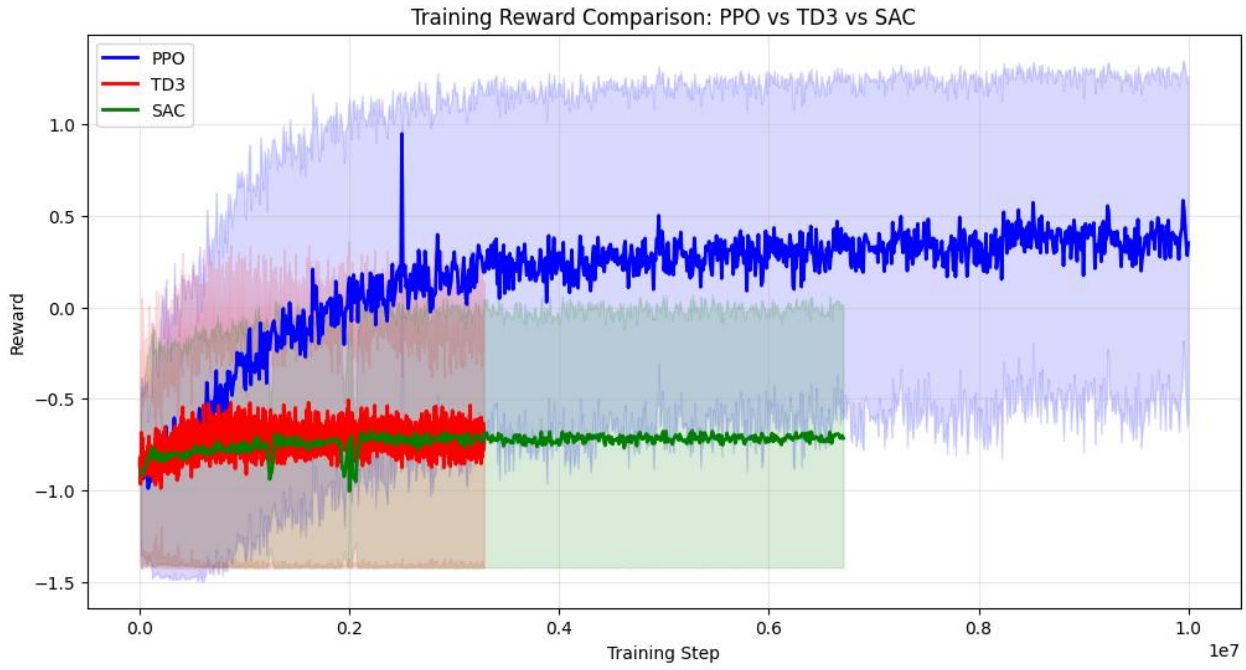


Figure 2 displaying training results of PPO, TD3, Soft Actor Critic

Model	Average Reward in training	Success Rate (100 episodes)
Original PPO (baseline)	.497 ± .053	27%
TD3	-0.71 ± 0.06	0%
SAC	-0.70 ± 0.01	20%
Replicated PPO	0.39 ± 0.10	0%

Table 1 displaying results of training each agent

All three model's conducted training ranging 2.5 – 10 million episodes. TD3's training was cut short due to its lack of convergence and need to train the sac model. TD3 and SAC had very similar training results of -.71 and -.7 respectively, their results are nearly identical in training growth. PPO was the only model to properly converge during training with an average reward of .39. During testing, the original provided PPO in the repo and the SAC model where the only agents able to properly identify that target speaker at all. While in the rendered testing environment, the original PPO spent majority of the time shaking in place, taking upwards of 1-2 minutes to complete an episode. TD3 and the Replicated PPO both stayed in place with a constant jitter. SAC mainly picked simple directions and moved as fast as possible. Most successful runs by the sac were in the same direction regardless of where the correct voice was with only slight adjustments to reach it.

Discussion

The outcomes of this experiment were far from expectations. Firstly, in the original paper, it claims to have a success rate of 97 percent per 100 episodes, yet during testing in their

environment with no change it only achieved 27 percent. I suspect this could be caused by the effects of training in a headless environment and more specifically utilizing speedup while in it, however they do not elaborate the methods that they chose during training. The goal of using speedup was to increase delta time during training simulations to get the agent to complete episodes at a far greater pace, however when increasing delta time, the physics within the simulation stay at a standard speed. This process causes the number of sound samples for the agent to be received to be far lower since the output of sound can't match the speed of the agent. This issue could explain part of why all the models struggled the way they did. Future testing should be in their same headless states to properly evaluate the model's efficiency.

With this issue however, it begs the question of why the sac was the only headless trained model during my experiment that properly showed some resemblance of movement. I believe this can be attributed to the specific stochastic policy of sac treating this variation as mere noise. PPO (while stochastic) can completely fall apart when its actions and observations don't match. TD3 also should completely collapse in this situation since the deterministic policy needs every state action pair to have the same result which is hindered by this imbalance.

Another important aspect to observe is why the TD3 and SAC could not converge during training. I believe this can be attributed to the use of a replay buffer, notably having a replay buffer of 500,000 can significantly hinder the model's ability to explore. Both models exhibited being locked in local maxima during training stalling growth after a million episodes.

Overall, the results of this experiment are inconclusive to determining if the replay buffer can help the model's ability to navigate the audio environment. This is mainly because there are far too many variables that need to be corrected for a fair evaluation.

Future Work

For future work, the most important thing would be to replicate the experiment having the models use the exact same environment for training and testing, as well as avoiding time speedup if in unity. However, in training, it would take weeks to fully train a model without any speedup. With this issue I believe it would be valuable to re-create the experiment on a different platform outside of unity, specifically one where speedup can also speed up the physics of the simulation as well.

Beyond proper replication of the experiment, adjusting the time reward to also rely on length of movement to ensure that each action by the agent yields a legitimate translation in their environment (prevent agents just jittering in place)

Challenges

Trying to build is Baselines 3

There were many struggles I ran into trying to get the model to work in unity. Because of this I eventually tried wrapping it into gym so that I could easily implement the TD3 model. While this implementation was a lot easier, it caused significantly increased overhead during

training. Under circumstances where the environment was less taxing, standard baselines 3 is a better option for experimenting with unity in the future.

Enabling GPU acceleration

During development of the experiment, I needed to be able to accelerate training using GPU acceleration, however the version of unity the original project had did not support proper PyTorch acceleration. This led to me having to modify the navMesh of the engine to accept the older environment of the experiment.

Debugging TD3 model

I learned much about how to debug a model during runtime. I had many issues where I could not see what was causing the model to collapse, or if the model was getting stuck, so I learned how to produce debug logs from the unity ml library.

Conclusion

When reflecting on this experiment, I learned so much about so many different aspects of reinforcement learning. I learned how to build models in unique & uncomfortable environments as well as properly critiquing and diagnosing shortcomings of past research and looking for way to fill in those gaps. I definitely look forward to continuing work on this paper as I've learned a lot about researching in general.

Sources

[1]Giannakopoulos, P., Pikrakis, A., & Cotronis, Y. (2021). *A deep reinforcement learning approach for audio-based navigation and audio source localization in multi-speaker environments.* arXiv preprint arXiv:2110.12778

[2]Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). *Proximal policy optimization algorithms.* arXiv preprint arXiv:1707.06347

[3]Haarnoja, T., Zhou, A., Hartikainen, K., Tucker, G., Ha, S., Tan, J., Kumar, V., Zhu, H., Gupta, A., Abbeel, P., & Levine, S. (2018). *Soft actor-critic algorithms and applications.* arXiv preprint arXiv:1812.05905

[4]Fujimoto, S., van Hoof, H., & Meger, D. (2018). *Addressing function approximation error in actor-critic methods*. arXiv preprint arXiv:1802.09477.