RL Project Report: Autopilot

Aditya Godavarthi[1], Jesudas Joseph Lobo[2], Surya Savarnika Pantina[3], Pravin Subramanian[4], Suhaibur Rehman[5], and Vineeth Kumar Arumgathilagar[6]

[1,2,3,4,5]Department of Computer Science , University of Bath, Bath, BA2 7AY
[6]Department of Electronic and Electrical Engineering , University of Bath, Bath, BA2 7AY
ag2919@bath.ac.uk[1] jjl77@bath.ac.uk[2] ssp65@bath.ac.uk[3]
ps2169@bath.ac.uk[4] sr2367@bath.ac.uk[5] vka28@bath.ac.uk[6]

1 Problem Definition

Autopilot, an AI flying game made in the Unity environment, was selected as the problem statement. In this game, we create a reinforcement learning agent that can fly about the environment following a predetermined course. On all four sides and on top, the environment is surrounded by boundaries. Checkpoints put along the flight route will direct the agent towards the finish line. The agent will get +0.5 point for successfully completing each checkpoint where as, if the aircraft takes longer than the normal time between checkpoints, it will be penalised by -0.5 points. Similarly, if the aircraft collides with any of the borders or the ground, it will incur a heavy penalty of -1 point. The project employs three sets of operations to run the flight: pitch change, yaw change, and enable/disable boost. The pitch change action causes the aircraft to descend nose down or rise with the nose up; a limit is placed on how far an agent may dive and rise in order to prevent the agent from flipping. The aircraft rotates side to side as a result of the yaw change. Finally, activating boost doubles the aircraft speed, allowing it to fly quicker in open places. To identify any obstacle, the agent observes ray casts produced from the aircraft nose in three directions, a total of 28*3. Aside from that, the RL agent monitors the aircraft's velocity, the distance from the aircraft to the next checkpoint on the route, and the rotation of the checkpoint, which totals 3+3+3 values. As a result, the agent makes a total of 93 observations at each decision step. When the agent crashes or the aircraft spends longer time than the maximum time steps without reaching the finish checkpoint, the episode finishes. As a result, our goal is to train the aircraft agent to fly through the checkpoints established in the course and reach the finish line while receiving the highest potential rewards.

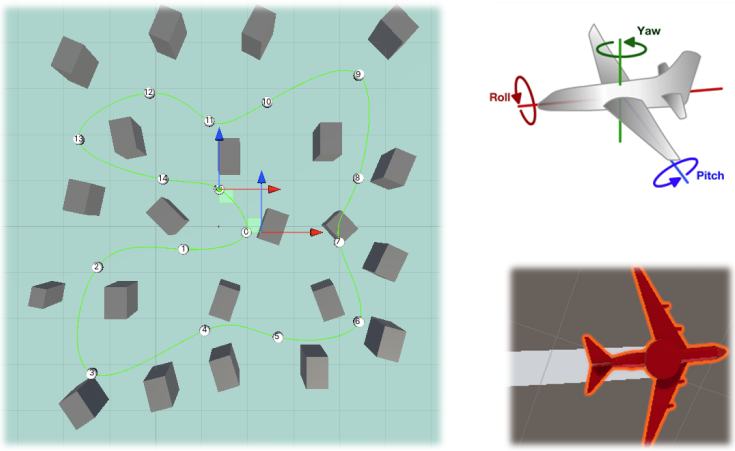

Figure 1: Environment and Aircraft Agent

2 Background

Motivated by the use of deep reinforcement learning techniques in various Unity and Atari games, we decided to investigate the various reinforcement learning techniques that would be effective in solving our problem. We investigated the following deep reinforcement learning variants, how they work, and the results obtained when applied to a similar type of problem:

A. Deep Q-Network (DQN and its extension)

Deep Q-learning is a model-free reinforcement learning technique that learns a Q-function that is used to determine the best action to take in a given state. Although a deep neural network is used to approximate this function, convergence issues may arise. (2017TimeVaryingFC) uses experience replay and reward clipping to stabilise multi-agent collaborative formation control from sensory input to improve on DQN. (Stanford (2016)) demonstrates the capabilities of a Multi-Agent Deep Q-Network called MADQN in a pursuit/evasion environment using a centralised training process. (Palmer et al. (2018)) used leniency to overcome outdated stored experience and discovered that Lenient-DQN (LDQN) models converge to optimal policy faster than the modified Hysteretic-DQN (HDQN) algorithm. By comparing Deep-RL results to expert human-level performance, the authors of (Mnih et al. (2015a)) demonstrated the power of odd DQN on Atari 2600 games. (Hasselt et al. (2016)) and (Wang et al. (2016)) improved on (Mnih et al. (2015a)) overestimations by applying the Double DQN and Dueling DQN models, respectively, and displayed increased performance. (Schaul et al. (2015)) extends the Double DQN algorithm with Prioritized Experience Replay, which adjusts the sampling distribution when there is a high prediction error. Deepmind merged six different techniques (Double-DQN, Prioritized Experience Replay, duelling networks, multi-step learning, distributional reinforcement learning, and noisy linear layer for exploration) into an agent dubbed Rainbow that outperformed the state of the art.

DQN has the advantage of resolving the convergence problem in value function estimation. It also trains more quickly and does not utilise ground truth data to train the Q-value estimator. The downside of DQN is that the Q-values are overestimated because one's network is used for both estimation and ground-truth data for the estimator.

B. Proximal Policy Optimization

To avoid excessively large weight updates, a method based on the Actor-Critic model cuts the loss function (which often lead to training instabilities). PPO is a simplified version of the prior TRPO (Trust Region Policy Optimization) method. (Schulman et al. (2017)) uses PPO against the Atari 2600 benchmarks to show that PPO outperforms typical online policy gradient approaches and is a good baseline for Deep-RL tasks because to its balance of sample complexity and algorithm simplicity. The authors of (Schulman et al. (2015)) applied TRPO to the Atari 2600 benchmark tests and discovered that it leads in monotonic improvements when compared to human and baseline DQN results. The authors of (Gupta et al. (2017)) extended this work by simulating cooperative multi-agent control in the pursuit/evasion game using a modified policy-scaled Trust Region Policy Optimization (PS-TRPO) method.

PPO has the advantage that the advantage function determines how good an action is in comparison to the average for a specific condition. It also takes less time as the environment's complexity grows.

C. Hierarchical Reinforcement Learning

According to (Hengst (2010)), HRL divides a reinforcement learning problem into a hierarchy of subproblems or sub-tasks so that higher-level parent tasks invoke lower-level child tasks as if they were primitive actions. There may be numerous levels of hierarchy in a decomposition. Hierarchical Reinforcement Learning has the advantage of being able to perform difficult tasks that DQN cannot, but it also has drawbacks such as sophisticated algorithms that are computationally expensive. Hierarchical reinforcement learning was utilised in the atari ping pong game (ttps://doi.org/10.48550/arxiv.1909.12465) and it was discovered that the agent was able to achieve human-level efficiency in only 0.25 times the amount of frames required by a human.

3 Method

We decided to put DQN into action and compare its performance with experience replay and PPO, Actor-Critic Style. PPO is a policy-based approach that directly searches the policy space for the

optimal policy, whereas DQN is a value-based method that approximates the optimal action-value function. We chose DQN because it overcomes the convergence problem in value function estimate. We chose PPO since it takes less time as the complexity of the environment rises. We did not employ HRL this time because it is a sophisticated method with a large computing footprint.

Algorithm 1: deep Q-learning with experience replay.

Initialize replay memory D to capacity N

Initialize action-value function Q with random weights θ

Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$

For episode = 1, M **do**

  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$

  For t = 1,T **do**

    With probability ε select a random action a_t

    otherwise select $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$

    Execute action a_t in emulator and observe reward r_t and image x_{t+1}

    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$

    Store transition $\left(\phi_t, a_t, r_t, \phi_{t+1}\right)$ in D

    Sample random minibatch of transitions $\left(\phi_j, a_j, r_j, \phi_{j+1}\right)$ from D

    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}\left(\phi_{j+1}, a'; \theta^{-}\right) & \text{otherwise} \end{cases}$

    Perform a gradient descent step on $\left(y_j - Q\left(\phi_j, a_j; \theta\right)\right)^2$ with respect to the network parameters θ

    Every C steps reset $\hat{Q} = Q$

  End For

End For

Figure 2: DQN pseudo-code, Source: (Mnih et al. (2015b))

A. DQN with experience replay

Below figure depicts the implementation of the DQN algorithm. Before we begin training, we first configure the replay memory data set D to capacity N. As a result, the replay memory D will store N total experiences. The network is then initialized using random weights. Following that, we initialize the episode's starting state for each episode.

Now, for each time step t inside the episode, we either explore the environment and choose a random action, or we exploit the environment and choose the greedy action with the highest Q-value for the given state. The selected action a_t is then executed in an emulator.

So, if the selected action was to go right, the agent would move right from an emulator where the actions were being done in the actual game world. The reward r_{t+1} for this action is then seen, as is the next state of the environment s_{t+1}. The whole experience tuple $e_t = (s_t, a_t, r_{t+1}, s_{t+1})$ is then saved in replay memory D.

B. Proximal Policy Optimization, Actor-critic Style

The Actor-Critical approach is Proximal Policy Optimization (PPO). The Actor-Critic system, as the name implies, includes two models: the Actor and the Critic. The Actor relates to the policy and is used to select the agent's activity and update the policy network. The Critic is equivalent to the value function Q(s,a) (for action value) or V(s) (for state value). The Critic modifies the network parameters for the value function utilized during the Actor update. The actor-network takes an observation (state) as input and returns a list of probabilities, one for each action. These probabilities create a distribution, from which the action can be chosen via sampling. The critic network also gets the state as input and produces a single integer indicating the estimated state value of that state to represent the state value function.

Algorithm 1 PPO, Actor-Critic Style

for iteration=$1, 2, \ldots$ **do**
  for actor=$1, 2, \ldots, N$ **do**
    Run policy $\pi_{\theta_{\text{old}}}$ in environment for T timesteps
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
  end for
  Optimize surrogate L wrt θ, with K epochs and minibatch size $M \leq NT$
  $\theta_{\text{old}} \leftarrow \theta$
end for

Figure 3: PPO pseudo-code, Source: (Lim et al. (2020))

C. Curriculum Learning

Curriculum learning (CL) is a training approach that simulates the relevant learning sequence in human curricula by training a machine learning model from easier data to tougher data Wang et al. (2021). In this method of training a machine learning model that progressively introduces to increasingly challenging components of a problem such that the model is constantly optimally challenged Bengio et al. (2009). This concept has been around for a long time, and it is how all humans learn. There is an ordering of courses and subjects in any kid basic school education. For example, arithmetic is introduced before algebra. Similarly, algebra is studied prior to calculus. The previous topics' abilities and knowledge serve as a foundation for subsequent classes. The similar approach may be used to machine learning, where training on easy tasks can serve as a scaffolding for future tougher jobs.

4 Results

The aircraft agent has been trained using DQN with experience replay and the following results were noticed.

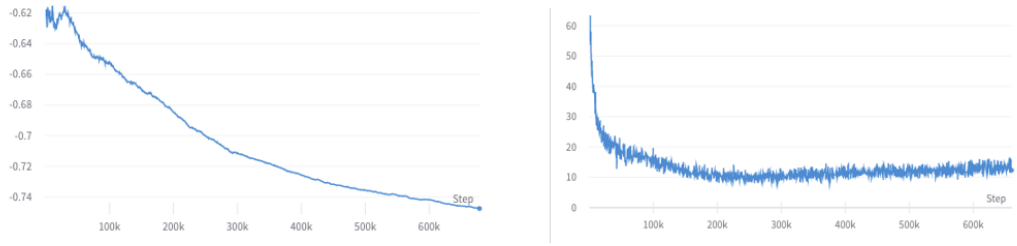

Figure 4: Average Rewards, Loss with DQN

After considering 16 agents and training them simultaneously with the same algorithm, the following results were obtained:

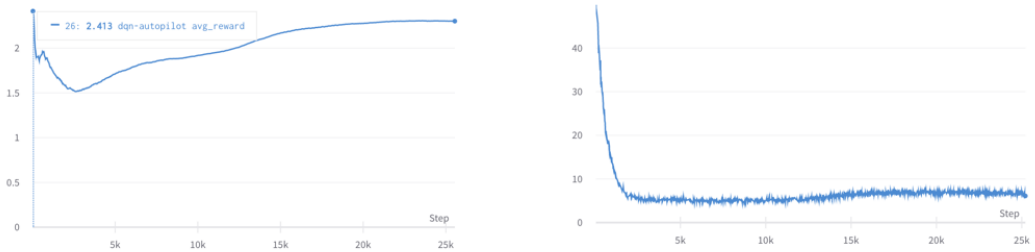

Figure 5: Average Rewards, Loss with DQN with 16 agents

We have implemented PPO for a single agent and these are the average rewards and loss observed:

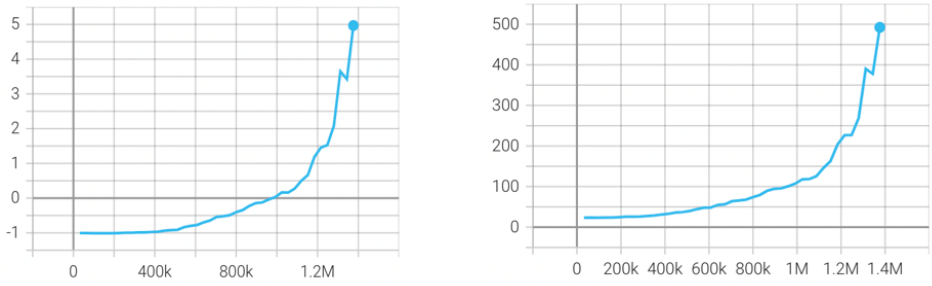

Figure 6: Cumulative Rewards and Episode Length with PPO

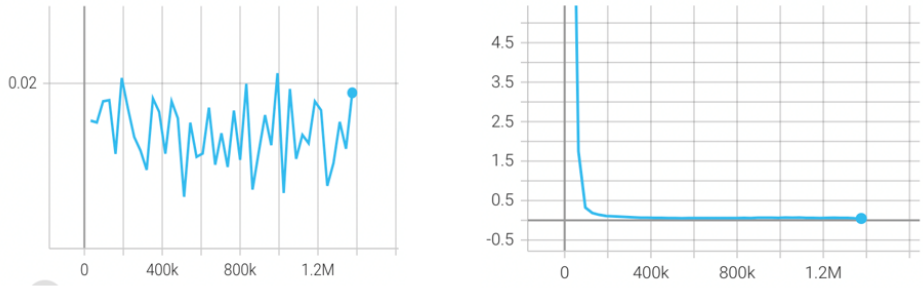

Figure 7: Policy Loss and Value Loss with PPO

Though the average rewards attained by the agent has increased with PPO, the training time taken is too long. To decrease the training time, we have introduced curriculum learning as well to guide the agent towards the checkpoints. The following are the results after introducing curriculum learning:

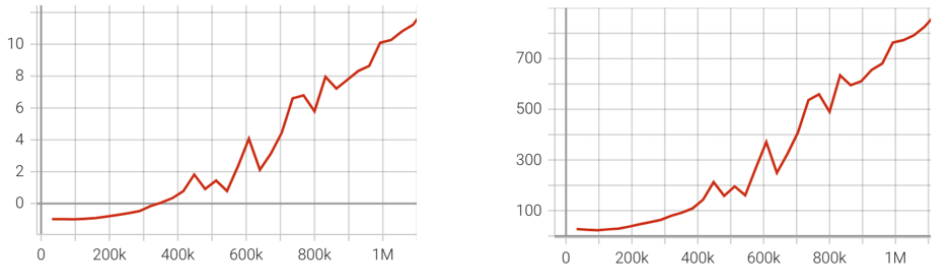

Figure 8: Cumulative Rewards and Episode Length with PPO and Curriculum Learning

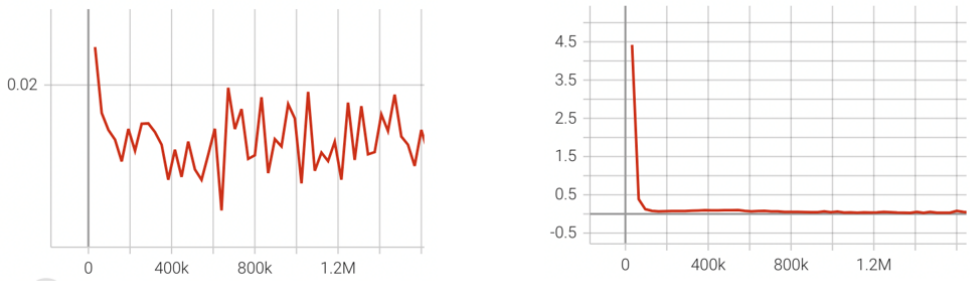

Figure 9: Policy Loss and Value Loss with PPO and Curriculum Learning

The comparison of the agent's performance using PPO without curriculum learning and with curriculum learning is shown in the figure below:

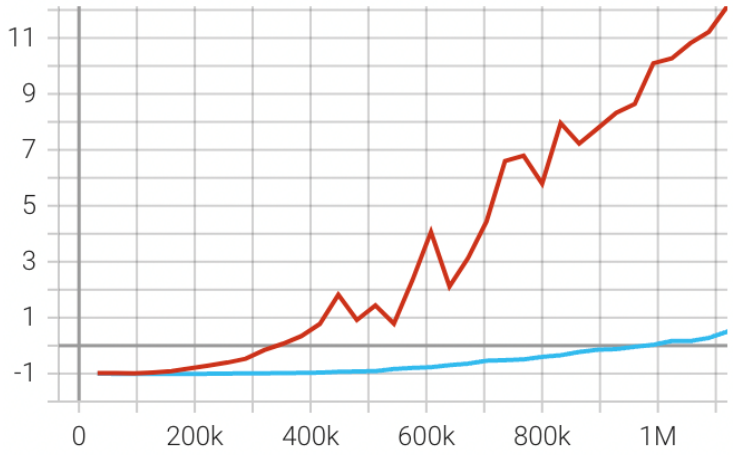

Figure 10: Comparison of Rewards

5 Discussion

Initially, the agent was trained in the chosen environment using DQN algorithm with experience replay. As the problem under consideration has a multi-action space, we have implemented multi-layer branching at the output Tavakoli et al. (2018). Agent was receiving negative rewards continuously as a positive reward is obtained only when it passes through the checkpoints. Hence, we have an issue of sparse reward, where we have extremely few positive prizes in the game compared to negative rewards. We have observed that though the loss is decreasing substantially with 50K number of iterations, there is no significant increase in the average rewards obtained by the aircraft after training it for a long time. The training time taken for the aircraft to reach the first checkpoint is very high even after 700K iterations. The cumulative rewards value after training for 700K steps is -0.75.

We increased the number of agents to 4 and replicated the environment 4 times, thereby, making the number of agents count to 16. The 16 agents were trained simultaneously with the same algorithm. It was observed that there was a positive increase in the average rewards obtained. The loss converged after about 2.5K iterations and the average rewards increase to 2.4 after 25K iterations as compared to the negative rewards with single after the same number of iterations. Though the convergence with respect to loss was much faster than that of a single agent, the overall performance was not quite as expected as it was computationally intensive to train 16 agents and the time taken was very high.

As DQN did not yield satisfactory results, we have implemented a policy-based algorithm, PPO, with a single agent. With the implementation of PPO, the average rewards attained by the agent has increased significantly with the number of steps than that of DQN. The cumulative rewards started seeing a positive trend after 900K steps and the loss converged after 200K steps. The training time taken to see a positive trend in the rewards as well as a decent performance of the agent was more than 3 hours. We have also tuned the hyperparameters used in the PPO algorithm to observe a faster convergence. We observed a positive trend in the rewards in about 2 hours of the training time.

As the training time for the aircraft to reach the checkpoint is very high due to the complexity of the environment, we have introduced curriculum learning Bengio et al. (2009) to direct our agent towards the checkpoints. A radius of 50 units has been set as the value of the curriculum towards the checkpoints. If the agent is under a radius of 50 units from the checkpoint, it is given a positive reward. This has reduced our training time drastically and we have observed a positive trend of rewards at about 300K steps. The performance of the agent was improved significantly.

Eventually, when the cumulative episodic rewards are satisfactory, the magnitude of the ra-

dius is decreased in steps from 30, 20, 10 and finally to 0, making it harder for the agent to attain more positive rewards. We have observed that during the transition of the magnitudes of the curriculum, there is a decrease in the episodic rewards. The loss convergence was obtained at about 200K steps and overall, we have observed a better performance of the agent within the complex environment.

6 Future Work

Currently, we have implemented curriculum learning to improve our agent's performance in the course-path by hard coding the values of the curriculum. This can be improved further by automating the curriculum learning process by reading the curriculum by a configuration file and updating the environment dynamically.

Imitation Learning Ciosek (2021) and Curiosity Learning Pathak et al. (2017) can also be considered to solve the sparse reward problem that is being considered in our current environment.

The training performance can be further improved by training multiple agents in multiple environments in parallel. This will lead to a faster training and convergence in the rewards earned.

We can also implement Transfer Learning Torrey and Shavlik (2010) to take our trained agent in the current known environment and deploy it in an unknown environment and record its performance.

7 Personal Experience

As the environment chosen is quite complex, we got the opportunity to explore several reinforcement learning algorithms and improvements that we weren't aware of previously. As a team, we did research on various methodologies to tackle this problem and sparse reward issue.

Initially, we have considered DQN as the optimal solution, but while implementing the algorithm, we had several issues as the problem under consideration has a multi-action space. We tackled this using multi-layer branching at the output, but the results were not as expected after training. Due to the complexity of the environment, the agent's performance with our implemented algorithms weren't quite satisfactory.

But the introduction of PPO gave us good results and it made us dive more to look out for algorithms or methods to improve the performance. We as a team, were amazed to see the depth of the problems, concepts and solutions involved. The learning curve involved in this project was huge and we got to know more about algorithms such as curriculum learning, imitation learning, etc.

References

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009). Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48.

Ciosek, K. (2021). Imitation learning by reinforcement learning. *arXiv preprint arXiv:2108.04763*.

Gupta, J. K., Egorov, M., and Kochenderfer, M. (2017). Cooperative multi-agent control using deep reinforcement learning. In Sukthankar, G. and Rodriguez-Aguilar, J. A., editors, *Autonomous Agents and Multiagent Systems*, pages 66–83, Cham. Springer International Publishing.

Hasselt, H. v., Guez, A., and Silver, D. (2016). Deep reinforcement learning with double q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, page 2094–2100. AAAI Press.

Hengst, B. (2010). *Hierarchical Reinforcement Learning*, pages 495–502. Springer US, Boston, MA.

Hub, G. (2022). ml-agents/python-api.md at release$_2 unity - technologies/ml - agents$.

Lim, H.-K., Kim, J.-B., Heo, J.-S., and Han, Y.-H. (2020). Federated reinforcement learning for training control policies on multiple iot devices. *Sensors*, 20:1359.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015a). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M. A., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015b). Human-level control through deep reinforcement learning. *Nature*, 518:529–533.

Palmer, G., Tuyls, K., Bloembergen, D., and Savani, R. (2018). Lenient multi-agent deep reinforcement learning. *ArXiv*, abs/1707.04402.

Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. (2017). Curiosity-driven exploration by self-supervised prediction. In *International conference on machine learning*, pages 2778–2787. PMLR.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2015). Prioritized experience replay.

Schulman, J., Levine, S., Moritz, P., Jordan, M., and Abbeel, P. (2015). Trust region policy optimization. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, page 1889–1897. JMLR.org.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms.

Stanford, M. E. (2016). Multi-agent deep reinforcement learning.

Tavakoli, A., Pardo, F., and Kormushev, P. (2018). Action branching architectures for deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

Torrey, L. and Shavlik, J. (2010). Transfer learning. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, pages 242–264. IGI global.

Wang, X., Chen, Y., and Zhu, W. (2021). A survey on curriculum learning. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Wang, Z., Schaul, T., Hessel, M., Van Hasselt, H., Lanctot, M., and De Freitas, N. (2016). Dueling network architectures for deep reinforcement learning. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, page 1995–2003. JMLR.org.

Appendices

The following appendices contains a detailed overview of our description of the problem domain,all experimental details and, hyperparameters tuning.

Appendix A: Problem Domain

Environment

To load a Unity environment from a built binary file, the file was put in the same directory as envs explained in detail in Hub (2022).

```
from mlagents_envs.environment
import UnityEnvironment
env = UnityEnvironment(file_name="name", seed=1, side_channels=[])
```

The name of the environment binary is file_name (located in the root directory of the python project).

The worker_id specifies which port should be used for communication with the environment.

The seed_value is used to generate random numbers throughout the training phase. Setting the seed in deterministic environments promotes reproducible testing by assuring that the environment and trainers use the same random seed.

side_channels enables data interaction with the Unity simulation that is unrelated to the reinforcement learning loop.

Problem description

The selected problem statement is an Autopilot. It is an AI flight game developed in the Unity environment. In this game, a user or an agent flies an aircraft in a race track with multiple checkpoints. The environment is surrounded with boundaries on all four sides and also on the top. The flight starts from the starting point and it passes through every check points. So, Our main goal is to train the agent that helps the flight to cross through maximum checkpoints and thus achieves a higher score with each game played.

States

There are 4 different states which records 93 values totally. Top, Bottom and Forward raycast records 28 values each. Then 3 values are observed for rotation of checkpoint, 3 values are observed to measure the distance from the check-point, and 3 points are observed for the velocity.

Reward

We reward the points for every attempts made by the aircraft. If the aircraft successfully passes through the checkpoint, a reward of 0.5 points will be provided. If the aircraft crashes/collides anywhere in the environment, then -1 point will be rewarded. If the aircraft takes more time in between the checkpoints to cross it, then -0.5 point is rewarded. The agent will get rewarded with -1/max (steps) with each time step.

Action

We use 3 set of actions such as Pitch, Yaw and Boost. Pitch consists of 3 different values (-1,0,1) to control the aircraft for side-to-side axis. Yaw consists of 3 different values (-1,0,1) to control the movement of aircraft's nose perpendicular to the wings. Boost consists of only 2 values (0,1) to boost the speed of the aircraft.

Appendix B: Hyperparameters

a. Change in Curriculum

After attaining positive rewards with a curriculum of 50 units, the curriculum is decreased in steps ranging from 30, 20, 10 and finally to 0. The following figures showcase the cumulative rewards of the same:

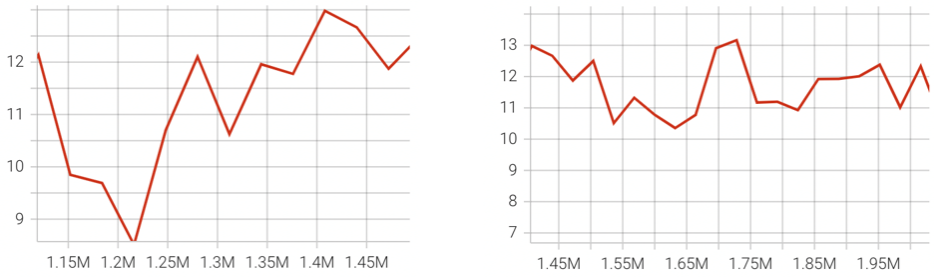

Figure 11: Cumulative Rewards with Curriculum values 30 and 20

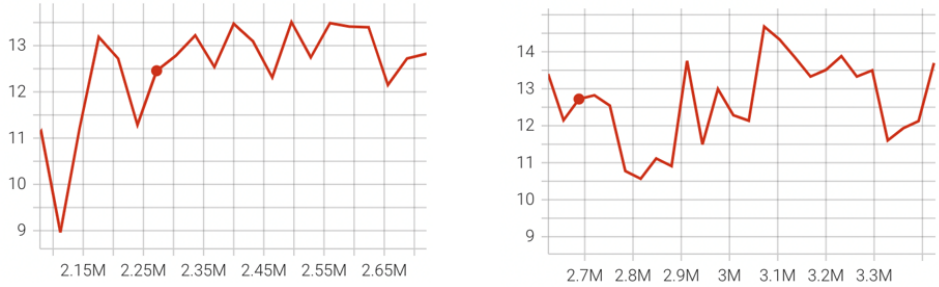

Figure 12: Cumulative Rewards with Curriculum values 10 and 0