Training DOOM and CartPole Agents Using REINFORCE, PPO, and DQN

Daniel Favour O. Oshidero
Department of Computer Science
University of Bath
Bath, BA2 7AY
dfoo20@bath.ac.uk

Siqi He
Department of Computer Science
University of Bath
Bath, BA2 7AY
sh3278@bath.ac.uk

Luis Eduardo D. A. Ballarati
Department of Computer Science
University of Bath
Bath, BA2 7AY
ledab20@bath.ac.uk

Yan Chun Yeung
Department of Computer Science
University of Bath
Bath, BA2 7AY
ycy55@bath.ac.uk

Ting-I Lei
Department of Computer Science
University of Bath
Bath, BA2 7AY
til27@bath.ac.uk

Yi-An Lin
Department of Computer Science
University of Bath
Bath, BA2 7AY
yal29@bath.ac.uk

1 Problem Definition

The initial problem defined for this project is to train a Deep Reinforcement Learning agent to play the "Defend the Center" (DFC) level in the VizDoom environment. In the level, the agent controls a first-person shooter stuck in the middle of a circular room with enemies approaching them from afar at from random locations spread across the 360 degrees landscape.

State space: First, a screen buffer representing the agent's first-person perspective of the current game scene (i.e., what they see in the circular room), returned as a numpy image array with 3 RGB channels. When passed through convolutional block, it delivers information on the relative positions of the enemies. Second, other game variables tracking the agent's "health", amount of remaining ammunition, whether they are alive, and the number of enemies killed.

Action Space: (0) Turn left, (1) Turn right, (2) Shoot.

Transition dynamics: The game operates in a deterministic environment, where each action has 100% probability of execution when selected, with exception of when the agent runs out of ammunition then the weapon would not be fired when the agent attempts to shoot (2).

Reward Function: The agent is rewarded +1 each time an enemy is killed. If the agent dies a death penalty of -1 is applied. An episode only ends when the agent dies. The goal is to accumulate as many rewards as possible with the amount of ammunition given. To do that the agent needs to kill each enemy before they become close enough to attack. Each time an enemy successfully attacks the agent, the "health" value would go down. The agent dies when their "health" is reduced to zero.

Additionally, we also tested and studied our models in the **CartPole-V1** environment, as the complexity of the VizDoom environment significantly blocked us from arriving at optimal training outcomes in the limited timeframe. A description of the CartPole environment configurations and problem definition can be found in Appendix D.

2 Background

Characterised by trial-and-error and delayed rewards, Reinforcement Learning (RL) is a machine learning technique closely mimicking the learning activity of a human brain (Sutton and Barto, 2018; Kalra and Patni, 2018). Traditional tabular RL is conditioned on a manageable state and action space, incorporating popular methods such as Q-learning, SARSA and Monte Carlo, all of which aim to discover exact optimal value functions (Sutton and Barto, 2018; Zhang, Zhang and Qiu, 2019; Nguyen, Nguyen and Nahavandi, 2020). However, to approach problems with high-dimensional state spaces like DOOM, Deep RL methods have proven to be a successful advancement in the field, where the function approximator for the optimal policy is a deep neural network (Mnih et al., 2015; Andriotis and Papakonstantinou, 2019).

The Deep RL landscape is underpinned by the Deep Q-Network (DQN) algorithm (Mnih et al., 2015), an off-policy and value-based approach that utilizes a deep neural network to estimate the optimal action-value function. It outputs the estimated value of each possible action and derives a policy by selecting the action that maximises the action value. In practice, however, the naïve DQN architecture is never implemented for many reasons. First, as the model is fed temporally correlated data, the agent is very likely to always prioritise recent data and "forget" what it observed a while ago, risking the loss of important information. Second, the network's weights can change drastically with small shifts in estimated action values, especially if ϵ-greedy policies are used to select actions, potentially causing turbulent training trajectories. Third, as the initial network is clueless of what values to expect for state and rewards, sudden appearance of unexpectedly large rewards can severely skew weights of the current network and destabilise training.

Thus, many variations of DQN have been designed to address such shortcomings. Deep Recurrent Q-network (DRQN), for instance replaces the fully connected layer right after the last convolutional layer of the policy network with a recurrent Long-Short-Term-Memory (LSTM). Double DQN (DDQN) decouples action selection and action evaluation to mitigate overestimation of action values (Van Hasselt, Guez and Silver, 2016). The Double Dueling DQN (DDDQN) integrates a dueling network architecture separating estimations of the q-value and the advantage with DDQN. Other enhancements such as experience replay and prioritised experienced replay (Lin, 1992; Schaul et al., 2015), gradient clipping (Zhang et al., 2019), and the replacement of squared loss with Huber Loss are often used in combination with DQN variations.

Meanwhile, policy-based (or policy-gradient) methods are underpinned by the Monte Carlo algorithm, searching for the optimal policy $\pi^*(a|s)$ directly in the policy space rather than learning optimal action-value functions as $Q^*(s, a)$ an intermediary. The basic idea is represented in its fundamental REINFORCE algorithm. More advanced methods aim to address problems of high variance and sensitivity to hyperparameter settings, which are prevalent in this family of models. The Trust Region Policy Optimisation (TRPO) for instance establishes a "trust region" constraint limiting how much policy parameters can change between updates (Schulman et al., 2015). Proximal Policy Optimization (PPO) (Schulman et al., 2017a) builds on TRPO, replacing the constrained optimisation with a clipped surrogate objective function and classical stochastic gradient descent. This not only penalises dramatic policy updates but also improves ease of implementation from a practical standpoint.

3 Method

3.1 Policy-Gradient: REINFORCE

As a fundamental policy-based method, REINFORCE (Willianms, 1988; Williams, 1992; Sutton et al., 1999) taking as input state-action pairs, computes the probability of choosing action a given state s and policy parameters θ. Optimization in policy-based methods is shaped by the objective function $J(\pi_{\boldsymbol{\theta}})$, which is the undiscounted expected returns. The gradient policy update is then performed using

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \nabla_{\boldsymbol{\theta}_t} J(\pi_{\boldsymbol{\theta}}),$$

such that the policy weights are updated in the direction for the agent to take actions that increase the expected return (Sehnke et al., 2010), where $\nabla_{\boldsymbol{\theta}_t} J(\pi_{\boldsymbol{\theta}})$ is the gradient of the objective function with regards to the current policy weight parameters. The policy gradient can be expressed by

$$\nabla_{\boldsymbol{\theta}} J(\pi_{\boldsymbol{\theta}}) = \frac{1}{|\mathcal{D}|} \sum_{\tau \in \mathcal{D}} \sum_{t=0}^{T} \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(A_t|S_t) G(\tau)$$

Where the policy function π_θ, $log\pi_\theta$ are accessible via the current policy network, with their derivatives accessed through backpropagation. The τ, a state-action trajectory in the form $\tau = (S_0, A_0, R_1, S_1, \ldots, S_{T-1}, A_{T-1}, R_T, S_T)$, and $G(\tau)$ represents the discounted return of a trajectory. Here, the $\mathcal{D} = \{\tau_0, \tau_1, \ldots, \tau_n\}$ represents a set of such trajectories, over which the policy gradient takes the sample mean to approximate expected future returns. The output of the network represents action preferences, which then undergoes a $Softmax$ transformation to derive action probabilities to perform action selection. The REINFORCE algorithm follows the structure of the optimisation steps outlined above, with the option to set a discounting factor γ such that the objective function is expressed as the discounted expected returns (Baxter and Bartlett, 1999).

Policy-gradient methods like REINFORCE tend to achieve better convergence, as the choice of action is directly shaped by the gradual updates of gradients. This helps to reduce the chance of dramatic changes to the agent's action choice caused by arbitrary changes in action value. Exploration is also in-built to the algorithm as actions are sampled from a set of non-zero probabilities derived from the outputs of the policy network. When applying REINFORCE to the Doom agent, we also introduce an entropy regulariser to encourage continued exploration as the rewards are generally sparse (only when the agent successfully kills an enemy), and thus risk becoming trapped in a local optimum (Williams and Peng, 1991; Ahmed et al., 2019).

3.2 Proximal Policy Optimisation (PPO)

A limitation of policy gradient methods is their direct implementations can cause high variance in the updates and become susceptible to local maxima. PPO was developed as an advanced variation of policy gradient methods. PPO adopts an actor-critic framework, alternating between sampling data through interaction with the environment, and optimising a "surrogate" objective function using stochastic gradient ascent (John et al., 2015; Schulman et al., 2017b). Unlike traditional policy gradient methods, the algorithm allows for multiple epochs of "mini-updates" from the same sample of data as a means of effectively utilising each sample for faster learning with fewer environment interactions. The key innovation of PPO to address high variance is the incorporation of a clipped probability ratio as a conservative approach towards estimating the performance of a policy during optimisation (Ratcliffe, Hofmann and Devlin, 2019; Schulman et al., 2017b). This helps to protect training from overly assuming improvements resulting from policy parameters change.

PPO updates a policy π_θ by optimising the *clipped surrogate objective function*:

$$L^{\text{CLIP}}(\boldsymbol{\theta}) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\boldsymbol{\theta})\hat{A}_t, \text{clip}(r_t(\boldsymbol{\theta}), 1-\epsilon, 1+\epsilon)\hat{A}_t \right) \right],$$

where $r_t(\boldsymbol{\theta}) = \frac{\pi_{\boldsymbol{\theta}}(a_t|s_t)}{\pi_{\boldsymbol{\theta}_{\text{old}}}(a_t|s_t)}$ is the probability ratio, and ϵ is the clipping range. Policy parameters $\boldsymbol{\theta}$ is updated through:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha \nabla_\theta L^{\text{CLIP}}(\boldsymbol{\theta}),$$

iteratively across training epochs until convergence, where α is the learning rate. $L^{\text{CLIP}}(\boldsymbol{\theta})$ is a combination of two distinct strategies. First, the probability ratio $r_t(\boldsymbol{\theta})$ describes the likelihood of choosing action a_t under the new policy π_θ compared to the old policy $\pi_{\boldsymbol{\theta}_{\text{old}}}$. This ratio is clipped between $1-\epsilon$ and $1+\epsilon$ to prevent drastic updates which can destabilize learning. $\hat{A}_t$ represents an advantage estimate at time t, known as *generalised advantage estimation*, which is used as an indicator of whether an action taken is better or worse than the policy's average, essentially directing the optimization towards more rewarding actions (Gu et al., 2021).

PPO has been demonstrated to outperform traditional policy gradient methods, providing balanced sample complexity, ease of implementation, and computational efficiency (Schulman et al., 2017b). Despite its strengths, PPO encounters challenges such as prematurely shrinking the exploration variance causing convergence to suboptimal policies, as well as high sensitivity to hyperparameter settings (Kobayashi, 2023; Fernand et al., 2021; Hämäläinen et al., 2020). To mitigate these, our methods attempted several enhancements: an entropy bonus was introduced to the objective function to encourage exploration and prevent early convergence (Boudlal, Khafaji and Elabbadi, 2024). Advanced hyperparameter tuning was also performed using a weighted average of tested values to select the most ideal parameters (Zhang et al., 2021). Additionally, a reasonable amount of action space noise was added, indirectly forcing the agent to continuously explore a wider set of actions to break the trap of local optimum (Hollenstein, Martius and Piater, 2024).

3.3 Deep Q-Network (DQN and DDDQN)

The primary concept of DQN is represented with the formula below (Zhu et al., 2024) where R_{t+1} is the reward at time t+1, $Q(S_{t+1}, a; \theta_t)$ is the q-value, and θ_t is denotes the policy network's parameters at time t.

$$Y_t^{DQN} = R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \theta_t)$$

In this project, we opted for the Double Dueling Deep Q-Network (DDDQN) as an extension of standard DQN, the structure of which is well-represented in the graph constructed by Zhu et al. (2024) (See Fig. 5 in Appendix B). "Double" refers to the decoupling of action selection from action evaluation, where a policy network chooses actions using the ϵ-greedy policy, and a separate target network calculating the resulting q-value (Nagy, Calliess and Zohren, 2023; Ausin, 2020). The separation allows the Policy network to update towards a semi-fixed yet steadily-moving target to stabilise training by reducing the correlation between q-value estimate functions and target value estimate functions. "Dueling", meanwhile, is achieved by splitting a network into two branches which separately estimate the state value function $V(s)$ of state s, and the advantage function $A(s, a)$. The two estimates are merged together as a single model output which are the estimated q-values (Nagy, Calliess and Zohren, 2023), like so:

$$Q(S, A, \omega, \alpha, \beta) = V(S, \omega, \alpha) + A(S, A, \omega, \beta),$$

Where α and β denote the parameters of the two model branches (Wang et al., 2016).

To better capture a sense of motion during a Doom episode, we stack game frames into stacks of fours before being passed into the convolutional layers and processed as image pixels (Albawi, Mohammed and Al-Zawi, 2017). An experience replay buffer was also adopted, such that previous observations were stored and sampled in a memory object for efficient usage of past experiences (Rizvi and Lin, 2019). Prioritised experience replay (PEP) via stochastic prioritisation was achieved using the TD error to help determine the probability of each experience being chosen into the sample batch, implemented using the binary *SumTree* data structure suggested by Simonini (2018). Each time an experience is sampled for training the agent, their TD errors are calculated and updated into the SumTree. When testing our models on CartPole agent, the effect of PEP was marked against uniform experience replay where all experiences have equal sampling probability.

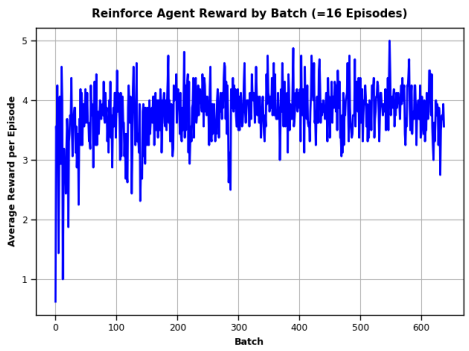
4 Results

Learning curves were plotted with the X-axis as training time and the Y-axis as average rewards earned in an episode. The computationally expensive nature of the Doom agent significantly prevented us from finding optimal configurations, with training sessions frequently requiring overnight runs. As a result, the Doom agents' performance did not converge to global optimum, with only slight improvements observed. However, to confirm our chosen Deep RL methods were correctly implemented and to better understand the individual model characteristics, the results of additional studies performed in the CartPole environment are presented and discussed in greater detail. The full list of hyperparameters for each run presented can be found in Appendix C.

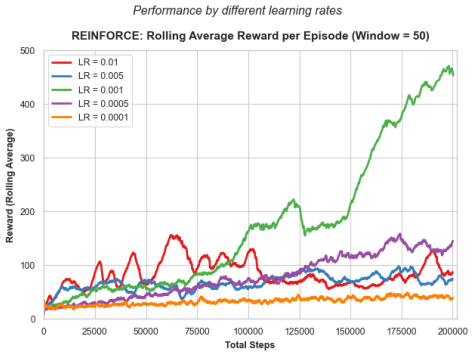
4.1 REINFORCE

Doom agent: The Doom agent was trained for over 20 hours by collecting full episode trajectories in batches of 16, with a learning rate of 0.0004 and a 0.001 entropy regulariser. Batch average rewards (See Fig. 1a) shows agent very quickly converged to an average of approximately 5 kills per episode after approximately 200 batches (3,200 episodes). From video replays we observed the agent had learned to efficiently explore its surroundings by continuously making left turns and attempting to shoot, but with weak attempts to aim and wasting ammunition. As training continued, no further significant improvement was observed for the next 12 hours.

CartPole agent: The CartPole agent, on the other hand, learned steadily over 200,000 time-steps when carrying a learning rate of 0.001, achieving a rolling average reward per episode of 200+ by 150,000 time-steps. The curves (see Fig. 1b) suggest the model is very sensitive to learning rates, where both higher and lower rates can significantly prevent learning.



(a) REINFORCE Doom Agent Reward Curve

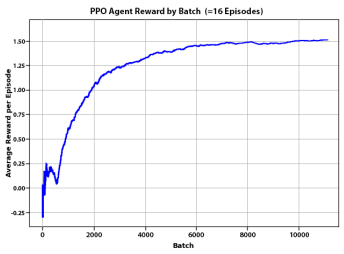


(b) REINFORCE CartPole Agent Rewards

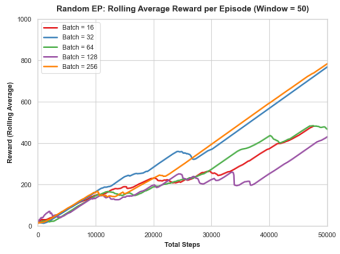
4.2 PPO

Doom agent: The Doom agent was trained with a learning rate of 0.0001, GAE lambda of 0.92, 0.3 entropy coefficient, and 0.5 noise coefficient. The training curve shows a sharp rise earned early in training, followed by a consistent increase indicating the agent picks up effective strategies (see Figure 2a). However, after around 4,000 batches rewards began to plateau and converge to an average of 1.5, indicating that the model may have reached performance limits given its current settings and that additional training was unlikely to yield further improvements. This rapid growth and plateau is common in environments where random exploration at the initial phase leads to quick termination. Observing the agent's video performance, we see that unlike the REINFORCE agent which adopted a passive self-defense strategy, the PPO agent has learned to pro-actively search for enemies to kill, despite aiming inaccurately and getting confused whenever multiple enemies are in sight.

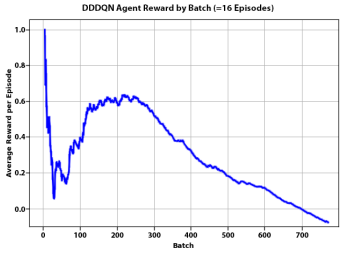
CartPole agent: PPO models were tested on the CartPole environment using several batch sizes, all showing steady increase in rewards over the course of 50,000 time steps and exceeding the 200 benchmark as seen in Fig. 2b. A batch size of 64 is identified as optimal for balancing learning speed and stability. Smaller batch sizes showed rapid initial learning but were prone to volatility, indicating large changes to model parameters, while larger batches demonstrate slower but more stable training. However, extended tests with batch size 64 reveals occasional occurrence of a significant performance drop around 40,000 time-steps (see Appendix E, 6). This behaviour is indicative of catastrophic forgetting, a phenomenon where the neural network abruptly discards previously learned data upon gaining new information, causing initially successful strategies to undergo rapid deterioration (Goodfellow et al., 2013).



(a) PPO Doom Agent Reward Curve



(b) PPO CartPole Agent Rewards



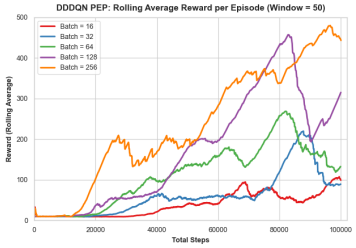
(c) DDDQN Doom Agent Reward

4.3 DDDQN

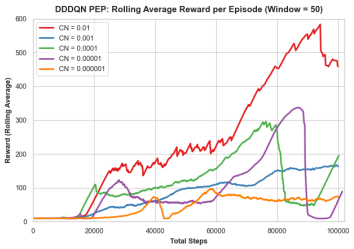
Doom agent: DDDQN with prioritised experience replay (PEP) is implemented on Doom agent with a learning rate of 0.00025 and epsilon decay range of 1.0 to 0.001, with a decay step of 0.00005. Rewards were averaged in batch of 16 episodes. After training for 230,000 time-steps, the plot (See Fig 2c) shows a moderate reward of 2 kills per episode in the early training, tanks dramatically, then displays slight improvements up until batch, 250 but fails to learn in later episodes. Among all models implemented for this project, DDDQN has the worst performance on the Doom agent, with

negligible improvements observed in the recordings. Severe time constraints meant we were not able to demonstrate the potentials of DDDQN in the Doom environment.

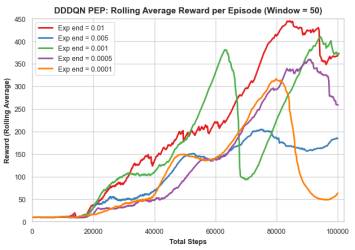
CartPole agent: DDDQN with prioritised experience replay (PEP) and random uniform experience replay (REP) were both implemented on CartPole, for 100,000 time-steps. Sampling batch sizes, clipping norm parameter, and the epsilon threshold were tested at various values (See Appendix C). As shown in Figs. 3a, 3c, 3b, 4b, 4a, 4c, whilst most agents were able to learn over various periods and solve the environment by exceeding 200 rewards, training was unstable in many cases, once more indicative of catastrophic forgetting. Adjusting the clipping norm parameter somewhat mitigated this, but lead training being significantly decelerated. Holding other parameters constant, PEP agents generally tended to learn faster with less local volatility, favouring smaller clipping norms and higher epsilon thresholds. Both PEP and REP agents seemed to prefer larger batch sizes.



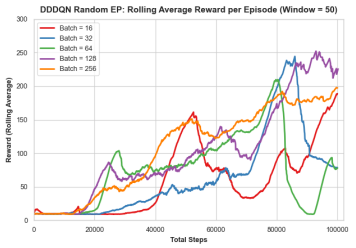
(a) DDDQN PEP Agent Rewards
At different Batch sizes



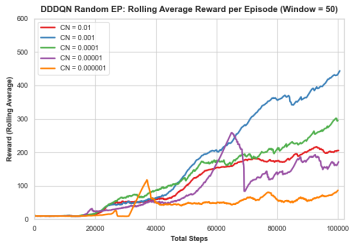
(b) DDDQN PEP Agent Rewards
At different Clip norms



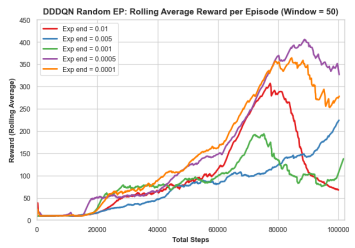
(c) DDDQN PEP Agent Rewards
At different Epsilon minimums



(a) DDDQN REP Agent Rewards
At different Batch sizes



(b) DDDQN REP Agent Rewards
At different Clip norms



(c) DDDQN REP Agent Rewards
At different Epsilon minimums

In addition to the models discussed, effort was invested over several days in the development of a Deep Recurring Q-Network (DRQN) model. Unfortunately, results could not be produced in time for both agents. For further insights, despite the unsuccessful outcomes, a detailed description of the DRQN model and the training results for the DRQN Doom agent is documented in Appendix F.

5 Discussion

As demonstrated in the test results and evaluations, the REINFORCE and PPO agents have both successfully picked up some survival behaviour. The REINFORCE Doom agent's reward curve attest to the intuition that direct policy-based methods tend to have a large training variance, as the average rewards per batch (of 16 episodes) have consistently displayed a large fluctuation range for an extended time period.[1] On the other hand, the PPO agent demonstrated a clearer advantage in the speed of learning, very quickly absorbing "skillful" tactics of rotating and stopping to shoot when identifying a enemy. This confirms the strength of PPO as an advanced policy-based method using the actor-critic framework and clipped surrogate objective function optimisation in making updates.

The relative strength of both policy-based algorithms extended to our implementations in the classic CartPole environment where the agents displayed much more stabilised learning trajectories than the DQN variant. The REINFORCE CartPole agent, in particular, achieved the best performance of all candidates, reaching almost 500 rewards per episode at 20,000 steps while still maintaining an upward trend. Majority of DDDQN agents displayed higher instability, with signs of catastrophic forgetting at various points in the 100,000 time-step horizon.

[1] In the original REINFORCE tutorial we followed, the agent took almost 2 days before breaking the average of 5 kills per episode and slowly increasing to 20 kills after 3+ days of training on cloud GPU (Hugh, 2022).

Overall, we can conclude that all three models were correctly implemented and improved the agents behaviour, with the exception of DDDQN having negligible effects on the Doom agent. Amongst the experiments we performed, policy-based methods showed an edge over value-based counterparts. We suspect that the use of the Double DQN structure requires more refined architectures and parameters settings, in order to coordinate the dynamics between target-setting in the target network and the optimisation mechanics in the policy network, such that the estimation of q-values as an intermediary does not inadvertently "block" efficient updates to the optimal policy.

6 Future Work

Despite witnessing progresses in both environments, we recognise great potentials of the agents, especially Doom, to improve further. Below are three techniques we would plan to explore in extended periods post submission.

Hyperparameter Tuning: Fine-tuning of hyperparameters is key in optimising the agents' training performance and stability. However, the tuning process is a generally computationally expensive procedure, especially when coping with high state spaces (Weerts, Mueller and Vanschoren, 2020). Different search algorithms could have been tried for hyperparameter tuning (Yu and Zhu, 2020), as well as leveraging existing open-source toolkits like Neural Network Intelligence (NNI) (Microsoft, 2021) or Katib (George et al., 2020) to assist, yet these toolkits may require a steep learning curve to set up and use effectively which we were not able to complete in the given timeframe.

Random Network Distillation (RND): RND is a new technique involving two neural networks, primarily used to encourage exploration in environments where rewards are sparse or non-obvious (Burda et al., 2018). By using the difference between the "Target" network and the "Predictor" network, the agent is guided to explore new strategies and new areas of the state space, reduce the risk of overfitting and thus lead to better outcomes that we are expecting. RND could be a potential enhancement to the DDDQN networks used to train the Doom agent.

Multiagent Deep RL (MADRL): Incorporating multiple agents within a single environment enables problem-solving of complex tasks through the communication of individual agents (Nguyen, Nguyen and Nahavandi, 2020; Gronauer and Diepold, 2022). An intricate interplay between agents may occur such that agents start to collaborate or act competitively to excel each other, depending on given tasks (Gronauer and Diepold, 2022). For instance, PPO-based multi-agent algorithms have shown strong performances with minimal hyperparameter tuning and without domain-specific algorithmic modifications (Yu et al., 2022). Through this approach, we would expect the Doom agent to achieve better performance and learn more effectively.

7 Personal Experience

As a team of conversion course students, we found the project to be the most challenging coursework this year. Many of us individually committed over 100 hours. As most members had no prior exposure to deep learning, much effort was dedicated to understanding source code and adapting it to the environments. The choice of VizDoom was initially inspired by a tutorial showcasing astonishing results using the StableBaselines library in a few hours time. As it turned out, this was a huge overestimation of both our computational power and our ability to fine-tune "raw" models built with PyTorch and TensorFlow. Specifically, we faced two major challenges: First, hyperparameter tuning was almost unrealistic due to a single run taking around half a day. Second, since we were uncertain whether we could directly use source code from GitHub, significant initial time was spent converting DQN source code from TensorFlow to PyTorch (which was a very fruitful learning experience). Despite the eventual success, the conversion consumed almost a week as it required us to grasp at various stages the shapes and types of data structures pass around in the model under both libraries.

Fortunately, this has been a tremendously rewarding journey, as we were pushed to dive deep into the models' fabrics and investigate reasons for sub-optimal training. Seeing the implementations performing on the CartPole environment reassured us that we achieved the objective of correctly understanding state-of-the-art Deep RL methods, refining them to complex environments like VizDoom will be the next-level progression to continuously work on. Some members of the team were so inspired by this experience that they now envision pursuing a career in RL research.

References

Ahmed, Z., Le Roux, N., Norouzi, M. and Schuurmans, D., 2019. Understanding the impact of entropy on policy optimization. *International conference on machine learning*. PMLR, pp.151–160.

Albawi, S., Mohammed, T.A. and Al-Zawi, S., 2017. Understanding of a convolutional neural network. *2017 international conference on engineering and technology (icet)* [Online]. pp.1–6. Available from: https://doi.org/10.1109/ICEngTechnol.2017.8308186.

Andriotis, C.P. and Papakonstantinou, K.G., 2019. Managing engineering systems with large state and action spaces through deep reinforcement learning. *Reliability engineering & system safety*, 191, p.106483.

Ausin, M.S., 2020. Introduction to reinforcement learning. part 4: Double dqn and dueling dqn [Online]. Available from: https://markelsanz14.medium.com/introduction-to-reinforcement-learning-part-4-double-dqn-and-dueling-dqn-b349c9a61ea1 [Accessed 2024-04-29].

Baxter, J. and Bartlett, P.L., 1999. *Direct gradient-based reinforcement learning: I. gradient estimation algorithms*. Citeseer.

Boudlal, A., Khafaji, A. and Elabbadi, J., 2024. Entropy adjustment by interpolation for exploration in proximal policy optimization (ppo). *Engineering applications of artificial intelligence* [Online], 133, 07, pp.108401–108401. Available from: https://doi.org/10.1016/j.engappai.2024.108401 [Accessed 2024-05-06].

Burda, Y., Edwards, H., Storkey, A. and Klimov, O., 2018. Exploration by random network distillation. 1810.12894.

Cartpole gym library [Online], n.d. Available online. Available from: https://www.gymlibrary.dev/environments/classic_control/cart_pole/ [Accessed 2024-05-07].

Fernand, R., Huang, S.Y., Ontañón, S. and Matsubara, T., 2021. An empirical investigation of early stopping optimizations in proximal policy optimization. *Ieee access* [Online], 9, 01, pp.117981–117992. Available from: https://doi.org/10.1109/access.2021.3106662 [Accessed 2023-08-07].

George, J., Gao, C., Liu, R., Liu, H.G., Tang, Y., Pydipaty, R. and Saha, A.K., 2020. A scalable and cloud-native hyperparameter tuning system. 2006.02085.

Goodfellow, I.J., Mirza, M., Xiao, D., Courville, A. and Bengio, Y., 2013. An empirical investigation of catastrophic forgetting in gradient-based neural networks. *arxiv preprint arxiv:1312.6211*.

Gronauer, S. and Diepold, K., 2022. Multi-agent deep reinforcement learning: a survey. *Artificial intelligence review*, 55(2), pp.895–943.

Gu, Y., Cheng, Y., Chen, C.L.P. and Wang, X., 2021. Proximal policy optimization with policy feedback. *Ieee transactions on systems, man, and cybernetics: Systems* [Online], pp.1–11. Available from: https://doi.org/10.1109/tsmc.2021.3098451 [Accessed 2022-03-24].

Hausknecht, M. and Stone, P., 2015. Deep recurrent q-learning for partially observable mdps. *2015 aaai fall symposium series*.

Hollenstein, J., Martius, G. and Piater, J., 2024. Colored noise in ppo: Improved exploration and performance through correlated action sampling. *Proceedings of the aaai conference on artificial intelligence* [Online], 38, 03, pp.12466–12472. Available from: https://doi.org/10.1609/aaai.v38i11.29139 [Accessed 2024-05-06].

Hugh, R., 2022. Vizdoom: Training defend the center for 75 hours [Online]. Available from: https://www.youtube.com/watch?v=5IqP9udOklw&list=PLdBvOJzNTtDUO4UC7R6N6_H-TFa78dka1&index=24 [Accessed 2024-05-01].

Hämäläinen, P., Babadi, A., Ma, X. and Lehtinen, J., 2020. Ppo-cma: Proximal policy optimization with covariance matrix adaptation. *International workshop on machine learning for signal processing* [Online], 09. Available from: https://doi.org/10.1109/mlsp49062.2020.9231618.

John, S., Moritz, P., Levine, S., Jordan, M.I. and Abbeel, P., 2015. High-dimensional continuous control using generalized advantage estimation. *arxiv (cornell university)* [Online], 06. Available from: https://doi.org/10.48550/arxiv.1506.02438.

Kalra, M. and Patni, J., 2018. Playing doom with deep reinforcement learning. *Recent trends in science, technology, management and social development*, p.42.

Kobayashi, T., 2023. Proximal policy optimization with adaptive threshold for symmetric relative density ratio. *Results in control and optimization* [Online], 10, 03, p.100192. Available from: https://doi.org/10.1016/j.rico.2022.100192 [Accessed 2023-03-28].

Lample, G. and Chaplot, D.S., 2017. Playing fps games with deep reinforcement learning. *Proceedings of the aaai conference on artificial intelligence*. vol. 31.

Lin, L.J., 1992. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8, pp.293–321.

Microsoft, 2021. *Neural Network Intelligence* (v.2.0) [Online], January. Available from: https://github.com/microsoft/nni.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G. et al., 2015. Human-level control through deep reinforcement learning. *nature*, 518(7540), pp.529–533.

Nagy, P., Calliess, J.P. and Zohren, S., 2023. Asynchronous deep double duelling q-learning for trading-signal execution in limit order book markets. *arxiv (cornell university)* [Online], 01. Available from: https://doi.org/10.48550/arxiv.2301.08688 [Accessed 2024-05-06].

Nguyen, T.T., Nguyen, N.D. and Nahavandi, S., 2020. Deep reinforcement learning for multiagent systems: A review of challenges, solutions, and applications. *Ieee transactions on cybernetics*, 50(9), pp.3826–3839.

Ratcliffe, D.S., Hofmann, K. and Devlin, S., 2019. Win or learn fast proximal policy optimisation. *2019 ieee conference on games (cog)* [Online], 08. Available from: https://doi.org/10.1109/cig.2019.8848100 [Accessed 2024-05-06].

Rizvi, S.A.A. and Lin, Z., 2019. engExperience replay–based output feedback q-learning scheme for optimal output tracking control of discrete-time linear systems. *International journal of adaptive control and signal processing*, 33(12), pp.1825–1842.

Schaul, T., Quan, J., Antonoglou, I. and Silver, D., 2015. Prioritized experience replay. *arxiv preprint arxiv:1511.05952*.

Schulman, J., Levine, S., Abbeel, P., Jordan, M. and Moritz, P., 2015. Trust region policy optimization. *International conference on machine learning*. PMLR, pp.1889–1897.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A. and Klimov, O., 2017a. Proximal policy optimization algorithms. *arxiv preprint arxiv:1707.06347*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A. and Klimov, O., 2017b. Proximal policy optimization algorithms [Online]. Available from: https://doi.org/10.48550/arXiv.1707.06347.

Sehnke, F., Osendorfer, C., Rückstieß, T., Graves, A., Peters, J. and Schmidhuber, J., 2010. Parameter-exploring policy gradients. *Neural networks* [Online], 23, 05, pp.551–559. Available from: https://doi.org/10.1016/j.neunet.2009.12.004 [Accessed 2020-09-20].

Simonini, T., 2018. $\mathrm{Deep}_r einforcement_l earning_c ourse/duelingdoubledqnwithperandfixed - qtargets/duelingdeepqlearningwithdoom(+doubledqnsandprioritizedexperiencereplay).ipynbatmastersimoninith$ [2024-05-05].

Sutton, R.S. and Barto, A.G., 2018. *Reinforcement learning: An introduction*. MIT press.

Sutton, R.S., McAllester, D., Singh, S. and Mansour, Y., 1999. Policy gradient methods for reinforcement learning with function approximation. In: S. Solla, T. Leen and K. Müller, eds. *Advances in neural information processing systems* [Online]. MIT Press, vol. 12. https://proceedings.neurips.cc/paper_files/paper/1999/file/464d828b85b0bed98e80ade0a5c43b0f-Paper.pdf.

Van Hasselt, H., Guez, A. and Silver, D., 2016. Deep reinforcement learning with double q-learning. *Proceedings of the aaai conference on artificial intelligence*. vol. 30.

Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M. and Freitas, N., 2016. Dueling network architectures for deep reinforcement learning. *International conference on machine learning*. PMLR, pp.1995–2003.

Weerts, H.J., Mueller, A.C. and Vanschoren, J., 2020. Importance of tuning hyperparameters of machine learning algorithms. *arxiv preprint arxiv:2007.07588*.

Williams, R.J., 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8, pp.229–256.

Williams, R.J. and Peng, J., 1991. Function optimization using connectionist reinforcement learning algorithms. *Connection science*, 3(3), pp.241–268.

Willianms, R., 1988. Toward a theory of reinforcement-learning connectionist systems. *Technical report*.

Yu, C., Velu, A., Vinitsky, E., Gao, J., Wang, Y., Bayen, A. and Wu, Y., 2022. The surprising effectiveness of ppo in cooperative multi-agent games. *Advances in neural information processing systems*, 35, pp.24611–24624.

Yu, T. and Zhu, H., 2020. Hyper-parameter optimization: A review of algorithms and applications. *Corr* [Online], abs/2003.05689. 2003.05689, https://arxiv.org/abs/2003.05689.

Zhang, B., Rajan, R., Pineda, L., Lambert, N., Biedenkapp, A., Chua, K., Hutter, F. and Calandra, R., 2021. On the importance of hyperparameter optimization for model-based reinforcement learning. In: A. Banerjee and K. Fukumizu, eds. *Proceedings of the 24th international conference on artificial intelligence and statistics* [Online]. PMLR, *Proceedings of machine learning research*, vol. 130, pp.4015–4023. https://proceedings.mlr.press/v130/zhang21n.html.

Zhang, J., He, T., Sra, S. and Jadbabaie, A., 2019. Why gradient clipping accelerates training: A theoretical justification for adaptivity. *arxiv preprint arxiv:1905.11881*.

Zhang, Z., Zhang, D. and Qiu, R.C., 2019. Deep reinforcement learning for power system applications: An overview. *Csee journal of power and energy systems*, 6(1), pp.213–225.

Zhu, Z., Chen, M., Zhu, C. and Zhu, Y., 2024. Effective defense strategies in network security using improved double dueling deep q-network. *Computers security* [Online], 136, p.103578. https://doi.org/https://doi.org/10.1016/j.cose.2023.103578.

Appendices

Appendix A: Submission Links

Link to agent performance video:	Video Link
Link to group presentation:	Presentation
Link to source code:	GitHub Repository

Appendix B: DDDQN Model Architecture

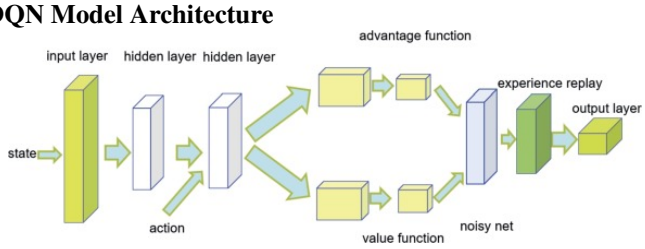


Figure 5: DDDQN structure (Zhu et al., 2024)

Appendix C: Full list of Model Hyper-parameters

The hyperparameter tuning for the CartPole tests was configured for REINFORCE, PPO and DDDQN models to evaluate combinations of the specified parameter ranges. Parameter values were manually changed for each run to analyse the effect values on training trajectory. This is reflected in the parameter tables below where a cell contains multiple values for certain hyperparameters.

REINFORCE Hyperparameters: Doom

Hyperparameter	Value
Learning Rate	0.0004
Batch size of collected episodes	16
Entropy regulariser	0.001

REINFORCE Hyperparameters: CartPole

Hyperparameter	Value
Learning Rate	0.01, 0.005, 0.001, 0.0005, 0.0001
Batch size of collected episodes	1

PPO Hyperparameters: Doom

Hyperparameter	Value
Learning Rate	0.0001
Discount Factor γ	0.90
Generalised Advantage Estimate (GAE) Lambda	0.92
Policy Clip	0.2
Entropy Coefficient	0.3
Noise Coefficient	0.5
Learning Batches	16
Batch Size	32
Number of Epochs	15

PPO Hyperparameters: CartPole

Hyperparameter	Value
Learning Rate	0.0005, 0.0003, 0.0001
Discount Factor γ	0.99
Generalised Advantage Estimate (GAE) Lambda	0.95
Policy Clip ϵ	0.1, 0.2
Entropy Coefficient	0.00001
Learning Batches	20
Batch Size	16 - 256
Number of Epochs	4

DDDQN PEP Hyperparameters: Doom

Hyperparameter	Value
Learning Rate	0.00025
Maximum τ	10,000
Initial/Maximum ϵ	1.0
Ending/Minimum ϵ	0.001
ϵ Decay	0.00005
Discount Factor γ	0.95
Pretrain Length	10
Memory Size	100,000
Replay Sample Size	64

DDDQN PEP & REP Hyperparameters: CartPole

Hyperparameter	Clip norm Tuning	Batch Size Tuning	Epsilon Threshold Tuning
Learning Rate	0.00001	0.00001	0.00001
Clip Norm	0.01, 0.001, le-4, le-5, le-6	0.0001	0.0001
Maximum τ	1,000	1,000	1,000
Initial/Maximum ϵ	0.1	0.1	0.1
Ending/Minimum ϵ	0.001	0.001	0.01, 0.005, le-4, 5e-4, le-4
ϵ Decay Steps	20,000	20,000	20,000
Discount Factor γ	0.99	0.99	0.99
Pretrain Length	1,000	1,000	1,000
Memory Size	50,000	50,000	50,000
Replay Sample Size	128	16, 32, 64, 128, 256	128

DRQN Hyperparameters: Doom

Hyperparameter	Value
Learning Rate	0.00025
Discount Factor γ	0.99
Initial/Maximum ϵ	1.0
Ending/Minimum ϵ	0.001
ϵ Decay	0.00005
Maximum τ	10,000
Batch Size	64
Maximum Steps per Episode	2,100
Pretrain Length	10
Memory Size	100,000

Appendix D: CartPole Environment Configurations

The CartPole environment is characterised by the following elements (CartPole Gym Library, n.d.).

Problem definition: A pole is placed vertically on top of a cart, the bottom of the pole is attached to the cart by an unactuated joint. The cart is placed along a frictionless track. At the start of an episode the pole is perpendicular to the surface of the cart. The pole has the tendency to fall due to gravity and once it tilts by more than 12 degrees it would fall and the episode ends. The goal is to stop the pole from falling by applying left and right forces to the cart. The environment is considered "solved" if the agent accumulates over 200 rewards in an episode.

State space: The state space is a Numpy array with shape (4,), comprised of four elements, listed below with their value min-max bounds.

1. Cart Position: $(-4.8, 4.8)$
2. incre $(-Inf, Inf)$
3. Pole Angle: $(-24$ degrees, 24 degrees), represented numerically as approximately$(-0.418, 0.418)$
4. Pole Angular Velocity: $(-Inf, Inf)$

Transition Dynamics: The game operates in a deterministic environment, where each action has 100% probability of execution when selected.

Action Space: (0) Push cart to the left, (1) Push cart to the right

Reward Function: A reward of +1 is given for every time-step the pole remains upright.

Appendix E: PPO Catastrophic Forgetting

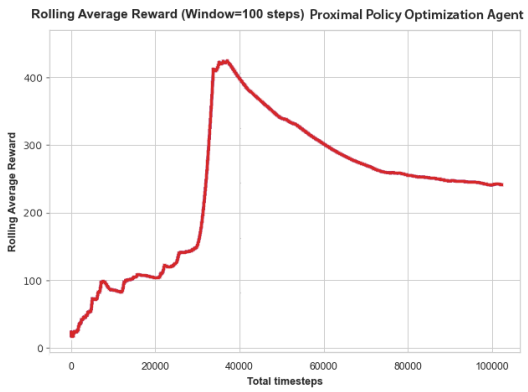


Figure 6: Batch Size 64: PPO Catastrophic Forgetting on CartPole

Appendix F: DRQN Model Description and Results on Doom Agent

Deep Recurrent Q-Networks (DRQN): Model Description

Reinforcement learning algorithms have improved significantly with the development of Deep Recurrent Q-Networks (DRQN) especially in solving the challenges of partially observable environments. DRQN improves the Q-function to consider not just the current observation but also the hidden state from the previous steps (Hausknecht and Stone, 2015). This improvement helps the networks to better capture the dynamics of the environment. Unlike DQN which action-value function based on full observations, DRQN integrates the concept of hidden states to adjust for the partial observation that exists in the environments (Lample and Chaplot, 2017). Although both DQN and DRQN use convolutional neural networks (CNNs) to process visual input, DRQN is more suitable for environments with partial visible because it includes recurrent neural networks (RNNs), the Long Short-Term Memory (LSTM) network, to help process the sequential data and hidden states, helping the agent make decisions based on past and present views. This together with the shared convolutional base in DRQN are powerful for extracting game features, and consequently speeding up training performance.

Deep Recurrent Q-Networks (DRQN): Performance on Doom Agent

The training curve shows that the mean episode length stabilises to just above 300 steps per episode after around 500 episodes of training. The mean episode reward converges close to 0, indicating that the agent has so far failed to learn survival strategies. This is likely to represent a shift from highly unpredictable untrained behaviour to repeatedly adopting ineffective strategies, settling into a local optima. Rewatching the recorded episodes at 500,000 time-steps, we observe that the agent has in fact 'learned' to shoot continuously without making left or right turns.

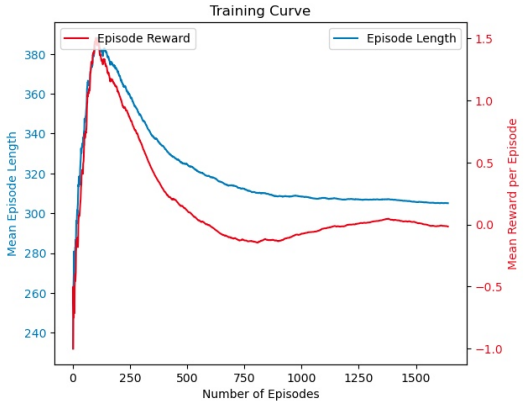


Figure 7: DRQN on Doom Agent Training Curve