Ayan Abhiranya Singh
CS 298

Navigating Classic Atari Games with Deep Learning

Project Advisor:

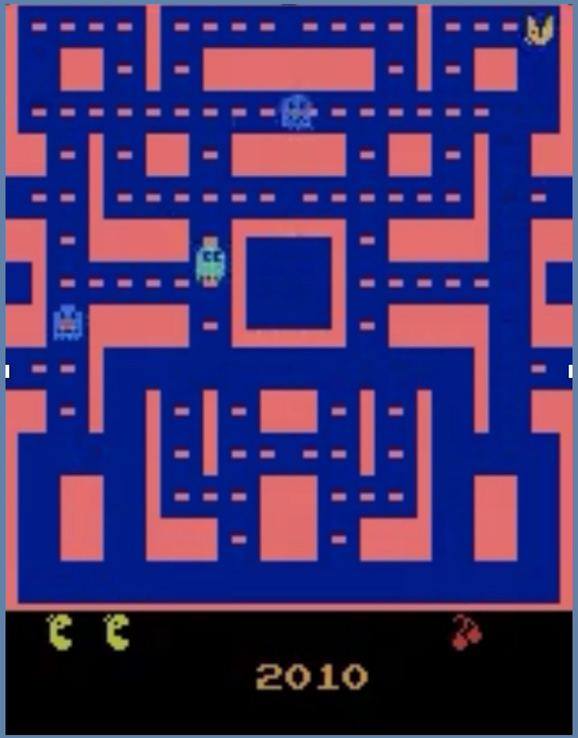
- Dr. Chris Pollett

Committee Members:

- Dr. Mark Stamp
- Dr. Kevin Smith

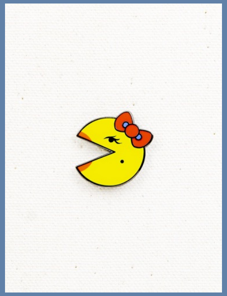
Outline

- Introduction
- Background
- Ms. PacMan
- Deep Q-Learning
- Experiments
- Conclusion



Introduction

- Project Goals:
  - Develop a reinforcement learning agent that is capable of playing Ms. PacMan.
  - Perform comparably, or beat, known benchmarks for RL agents for Ms. PacMan.
  - Modify the agent to perform up to a reasonable standard under conditions of latency.
  - Experiment with transfer learning with different mazes.



Background

- Reinforcement learning
  - Addresses decision-making problems that usually have some degree of uncertainty.
  - RL agents learn the best strategy for taking sequential decisions across time by interacting with a constantly changing environment.
  - A strategy to take repeated sequential decisions across time in a dynamic system is also called as a policy.
  - Generally, RL problems are structured within a framework that contains a state space, action space and a reward signal.

Background

- DQN (Deep Q Network)
  - Developed by DeepMind in 2013.
  - Able to reach superhuman performance on Space Invaders and Breakout.
  - Also able to provide good results on Pong, Seaquest, Beam Rider, Enduro and Q*bert.
  - Famously, learned (and mastered) a real human strategy in Breakout of breaking a portion of the wall and pushing the ball through to the other side. The ball then destroys the wall from the other side while the agent does nothing.
  - Struggles with games that tend to have longer trials and more of an exploration aspect, like Ms. PacMan.

DQN

- First deep learning model to learn directly from high-dimensional input.
- Variant on Q-Learning, an RL algorithm.
- Pivotal paper as it detailed a deep reinforcement learning model that solved a multitude of problems (games).
- Later research has built off this paper.

DQN playing Breakout and Lunar Lander

Breakout

Lunar Landar Environment
after 100 000 training timesteps
using the DQN RL algorithm

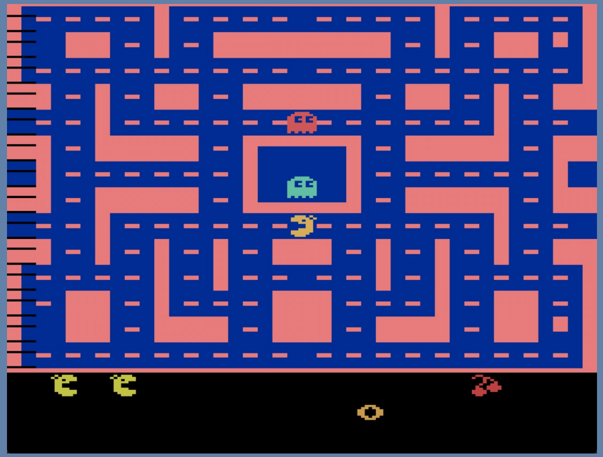
Lunar Lander

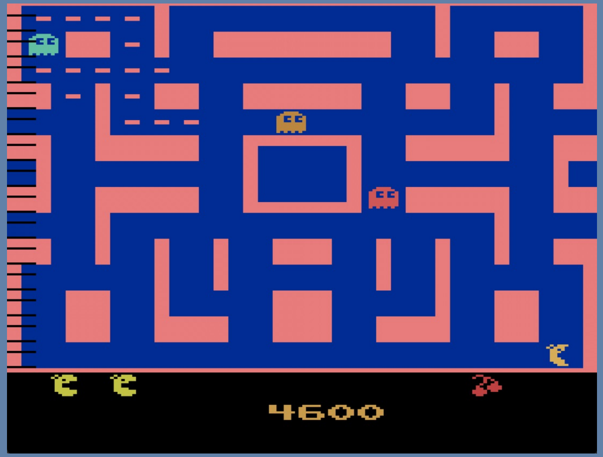
The credit-assignment problem

- Essentially, they developed the first deep learning model that was able to learn efficient control policies for classic Atari games from high-dimensional input, i.e., image frames from the game.
- Trials of these games (or episodes) usually run for many steps (sometimes to the order of tens of thousands of frames) and the reward returned, per-frame, often becomes very sparse.
- Credit assignment is a big challenge under this condition, i.e., learning what the right action should be given an observation.
- To solve this problem, Deep Q- Learning combines the Q-Learning algorithm with convolutional neural networks (CNNs).

Sparse rewards in Ms. PacMan

- Ms. PacMan is a game of exploration. The game involves collecting all the food pellets in the maze without the ghosts catching you.
- Over time, as more of the map is eaten away, the rewards become harder to find (the sparse rewards problem).
- Ultimately, as seen in the figure on the right, there may be no immediate reward available to the agent no matter the action it takes.
- However, the agent always prioritizes immediate rewards to speed up the learning process.





The Reward Maximization Problem

- Record observations and rewards at each time step.
- Q-Agent executes actions, which modify the environment and return an observation and a reward (which could be zero).
- We want to maximize that reward value at each time-step, t.
- Additionally, we want to discount future rewards with a discount factor, γ. Given an action a and state s at time t, the total future rewards from time t onwards, R, is given by the equation.
- $R_t = \sum_{t'=t}^{T} (\gamma^{t'-t} r_{t'})$

Deep Q-Learning

- This approach entails training a Q-function that accepts the state as input and outputs the predicted reward if action a is taken at state s. The function Q(s, a) is given by:
- $Q(s, a) \rightarrow R$
- Where R is the total future reward after an action a has been taken.

- It follows that a perfect Q-function will exactly predict the reward from a given time step onwards till the end of the episode.
- A neural network can output Q-values for all legal actions.

Online Q-Learning

- Since the perfect Q-function does not exist in practice, the Q-function for this neural network must be trained.
- Online reinforcement learning - given a state s, the Q-function can be estimated by simply playing through an episode and recording all the rewards.
- While the game is being played, the DNN is trained at the same time.
- Over time, if this training process is conducted iteratively, the Q-function will converge upon an optimal Q-function for the given environment.

Q-Function

- The Q-function models the Bellman recurrence equation. This means that the Q-function for a state s can be represented in terms of Q functions for states s’ and onwards.
- The star policy, $Q^*(s,a) \rightarrow\ R$, returns the highest Q-value possible given an action and state, and can be defined as:
- $Q^*(s,a) =\ \mathbb{E}_{s' \sim\, \epsilon}\,[\,r + \gamma \max_{a'} Q^*(s', a')\mid s, a]$
- Thus, we have a recurrence relation where the optimal policy $Q^*(s,a)$ may be frames in terms of $Q^*(s,a)$ for subsequent states. This recurrence relation lays the basis on which we can train the neural network.

Loss function

- The loss function for the neural network is defined as such:
  - $L_i(\theta_i) = \mathbb{E}_{s,a\sim\,\rho(.)}\,[\,\left(y_i - \,Q(s,a;\theta_i)\right)^2]$
- ρ(s,a) is a probability distribution over the states (s, a), known as the behavior distribution..

Loss function

- The "true" labels for the loss function:
- $y_i = \mathbb{E}_{s' \sim \epsilon} [\, r + \gamma \max_{a'} Q^*(s', a' \,;\, \theta_{i-1}) \mid s, a]$
- is the target for time step i .
- Thus, the DQN itself is used to compute both true and predicted labels to compute the squared loss in this deep reinforcement learning algorithm.

Model-Free

- This algorithm is model-free. This means that the algorithms itself simply learns a function that outputs a Q-value for each state-action tuple. There is no interaction with external components like the model or an emulator, to learn specifics about the reinforcement learning task.

Off~Policy

- The model learns what is known as the ε-greedy strategy:
- $a = \max_{a} Q(s, a;\ \theta)$
- The agent will randomly explore the environment with ε degree of probability and request Q-values from the DNN the remaining 1-ε amount of time.
- This helps mitigate exploitation of the neural network and also can speed up the training phase by reducing the number of calls to obtain Q-values.

Experience Replay

- Successive frames across the game’s environment are often extremely similar and correlated.
- Instead of using data samples as they are generated, a better approach to boost training performance is to store observations, actions and rewards in an “experience replay buffer” and samples those at random.
- Inherently, this implies some samples might never get selected for play, while others might get selected more than once. A good analogy for this is the process of randomly selecting or shuffling data prior training a model under supervised learning.

Experience Replay

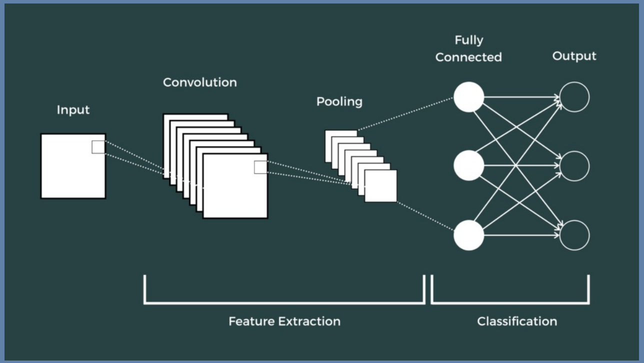
- The input state to the neural network comprises of a buffer of 4 frames (observations).
- Following the ε-greedy strategy from above, the agent either takes a random action with probability ε, or the model suggests the action to be taken that has the highest future reward.
- Once this action is executed, the next state of the game's environment is obtained, with a new observation and reward. This transition, from state s_t to state s_{t+1} is stored in the experience replay buffer.

Convolutional Neural Networks (CNNs)

- Convolutional neural networks (CNNs) are neural networks that work great with problems involving learning local structures, for example, object detection in images.
- The CNN consists of convolutional layers, which act as filter matrices of some set size, where the elements of the filters are the weights for that layer.
- These convolutional layers are usually followed by an activation function, which determines what the final output will be from one neuron in the network to the next, or, in some cases whether the neuron is on or off.

CNNs

- First convolutional layer might take an image as input. The filter applied on it may extract simple shapes like lines, diagonals, triangles.
- Subsequent convolutional layers will then be applied on the output from the previous convolutional layer The final layer of the CNN is fully connected and enables it to make classifications.
- The filter weights and biases of the CNN are learned as part of the training.



CNN Implementation

- 3 convolutional layers:
  - 32 filters of 8x8, stride of 4, activated by ReLU
  - 64 filters of 4x4, stride of 2, activated by ReLU
  - 64 filters of 3x3, stride of 1, activated by ReLU
- 2 dense layers:
  - 512 neurons, activated by ReLU
  - 5 neurons (size of the action space for the agent), with a linear activation function
- The model for this project was developed in Python and the Keras API in TensorFlow was used to implement the deep neural network described above. Game emulated via ALE & OpenAI Gym.
- Hardware: MacBook Air M1 8 GB of LPDDR4 memory. TensorFlow executes computations on the M1 GPU, which has 8 cores and supports the Apple Metal graphics optimization technology.

Experiment 1: Input Lag

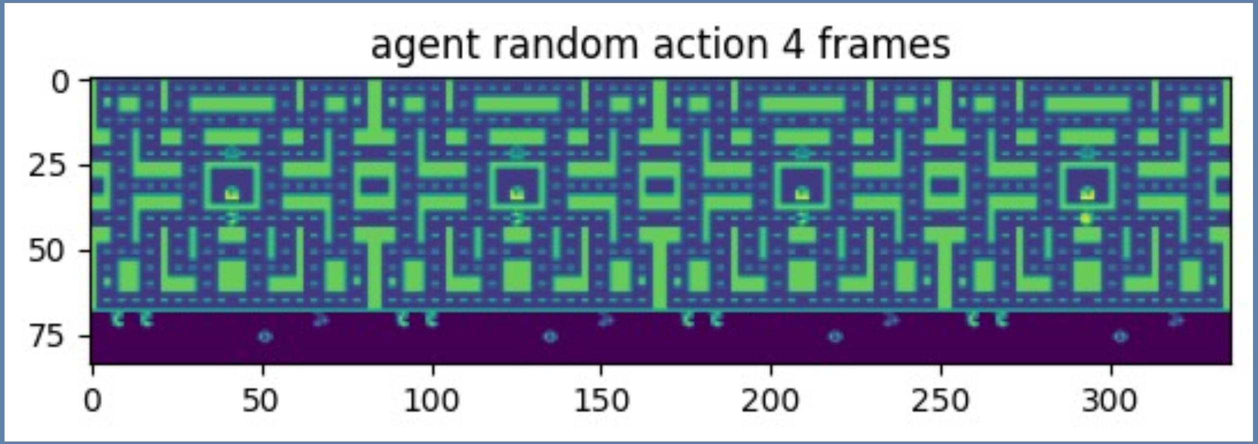
- One use case for an AI agent like this might be to deploy it over an internet connection to play similar games or even different mazes within Ms. PacMan.
- A setup like this could involve input lag with transfer learning.
- This would require some sort of exploratory analysis into packet loss and stuttering, which explains our motivations for the experiments in this section.
- To simulate lag within the program environment, we first need to experiment with the frames of Ms. PacMan running on the Arcade Learning Environment (ALE).

Experiment 1: Input Lag

- Measuring the time elapsed per frame is of particular importance in this experiment as that is the single control parameter that measures how much of lag we are inserting into the simulation.
- The approach is to time gameplay in the Stella Atari 2600 emulator (which is the emulator that powers the ALE rendering of Atari games) versus how many frames the agent in the Gym environment takes to move the same distance.
- One frame comes to represent approximately 0.085 seconds of gameplay. Thus, if we simulate a frame skip of 2 frames, we are effectively simulating a loss of 0.17 seconds of user gameplay.

Input buffer

- Input state buffer, after playing 1 move → [1112].



Experiment 1: Input Lag

- The baseline agent is trained over 5000 episodes, working on the reduced move set (NO-OP, UP, DOWN, LEFT, RIGHT).
- The agent scores 850 points. This sets a baseline model for the next experiment.
- Next, we duplicate frames for every pair of 2 moves. For example, input may look like [1111] → [1122] → [2233] → [3344], and so on.
- Under the condition of latency, the agent scores 530 points, a considerable drop off of 37.65% from the previous score of 850 which was obtained by the base model.
- This tells us that the agent needs training for more episodes and perhaps with a "gap" between frames as an adjustable training parameter.

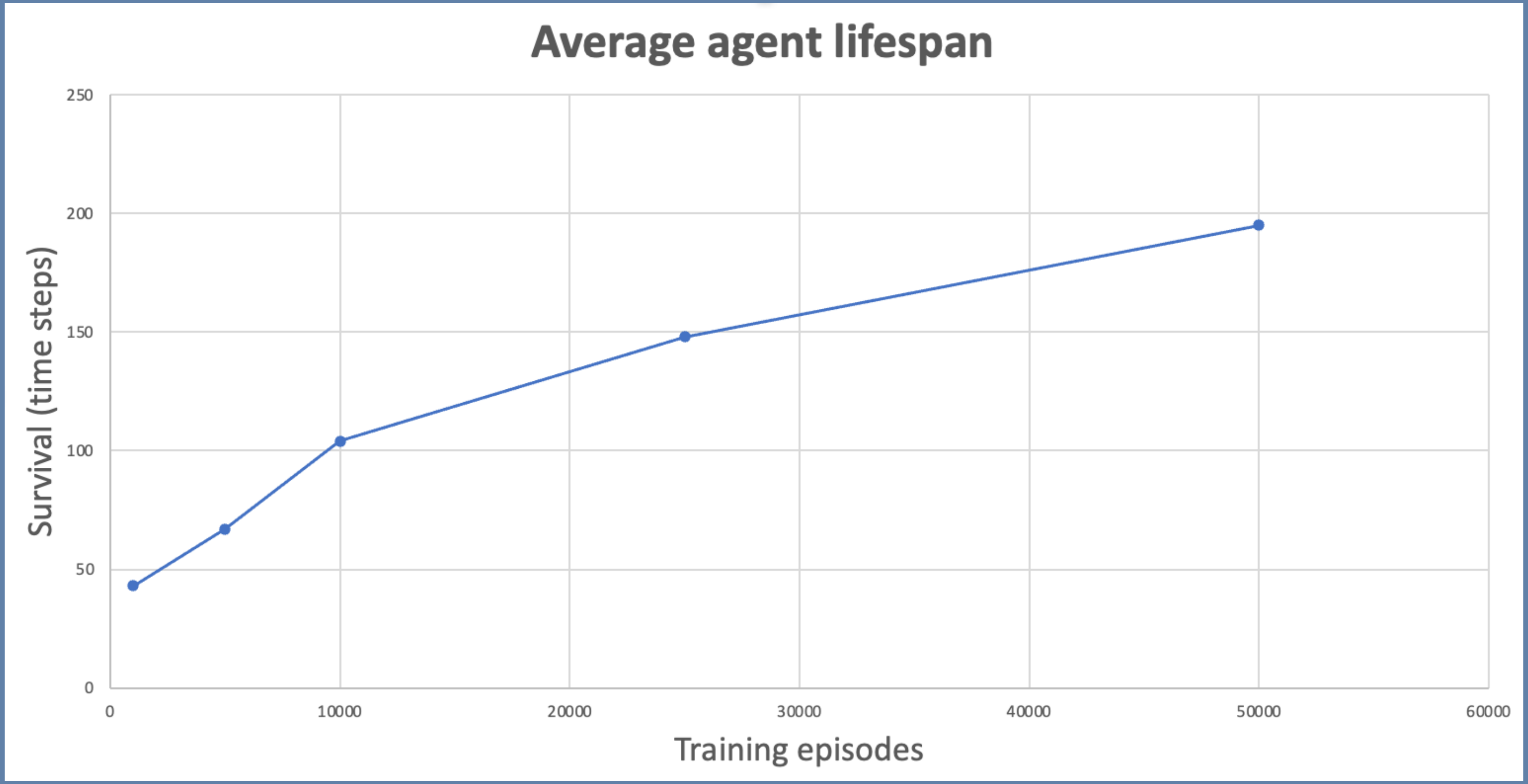
Experiment 2: Input Lag Modifiers

- Frame skipping is an important hyperparameter used in training the agent to play Atari games.
- The frame skip value represents the number of frames that should be "skipped", or ignored, between the agent's actions, during the training phase.
- Our motivation behind selecting a frame skip value:
  - Speeds up training the agent, since the time between frames is quite small.
  - Could potentially also improve performance with an input at an inconsistent frame rate.
- We select a frame-skip parameter of 4.

Experiment 2: Input Lag Modifiers

- An agent moving completely randomly scores an average of 307 points [18] per trial.
- This is the average we hope to beat over our experiments, as this would demonstrate an agent that is learning over time and playing better than simply moving randomly across the board.

Average agent lifespan



Experiment 2: Input Lag Modifiers

	3 Frame Cache With Coin-Toss	2 Frame Cache With Coin-Toss	8-Frame Input Buffer	4-Frame Input Buffer
Trial 1	470	510	1310	590
Trial 2	370	470	870	1740
Trial 3	810	440	730	700
Trial 4	990	470	350	970
Trial 5	540	470	600	440
Trial 6	450	660	710	1050
Trial 7	430	550	710	590
Trial 8	1140	520	1140	1300
Trial 9	770	520	750	740
Trial 10	530	310	870	900
Average	650	492	804	902
STDEV	262.7630957	88.6691729	269.0394436	389.0672607

Experiment 3: Optimizing Q-Function

- Through stochastic gradient descent, we aim to minimize the loss function over the iterations of the training process.
- The learning rate dictates how large of a “step” is taken towards the minimum. For the DQN training process, since we aim to learn an optimal Q-function, the gradient descent is minimizing loss on the Q-function.
- By changing the learning rate, we change the weight updates per iteration of the training process. The goal of the experiments in this section is to find an adequate learning rate to converge upon the optimal Q-function, thereby ensuring an efficient training process and high-performance agent.

Experiment 3: Optimizing Q-Function

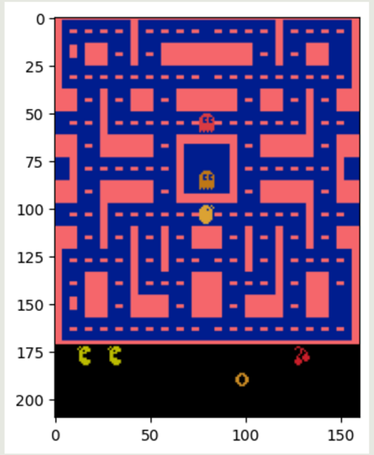
	5k eps, LR=0.001	5k eps, LR=0.0001	5k eps, LR=0.00005	10k eps, LR=0.0001	5k eps, LR=0.00025
Trial 1	330	550	70	390	1310
Trial 2	1820	400	70	990	870
Trial 3	370	950	70	940	730
Trial 4	330	550	70	480	350
Trial 5	330	680	70	490	600
Trial 6	370	800	70	420	710
Trial 7	330	850	70	990	710
Trial 8	340	560	70	520	1140
Trial 9	370	390	70	600	750
Trial 10	330	520	70	520	870
Average	492	625	70	634	804
STDEV	466.9713291	189.22356	0	241.3020238	269.0394436

Experiment 4: Frame-Diff

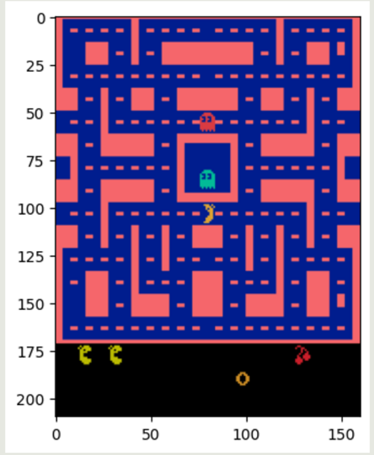
- The novel “Frame-Diff” approach involves building a quasi-attention technique for the CNN model by computing the difference between successive frames.
- By computing the difference between successive frames, we often obtain a result (due to the similarity between successive frames) where the frames only highlight what changed in the time lapse between them.
- States that are saved in the experience replay buffer consist only of the pixel values across frames that have changed – thereby reducing noise to the neural network.

Experiment 4: Frame-Diff

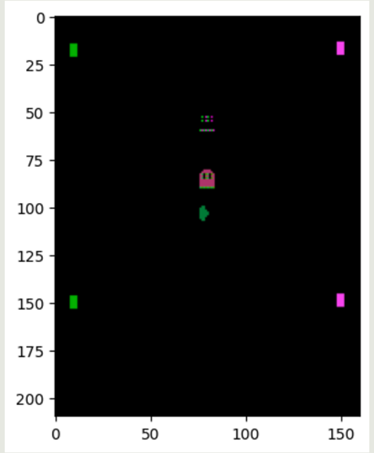
- Sample of a derived frame using Frame-Diff



- Frame 1 (F1)



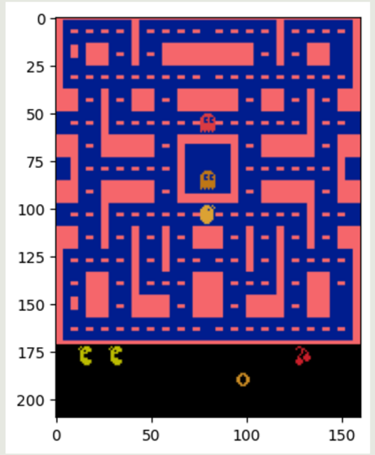
- Frame 2 (F2)



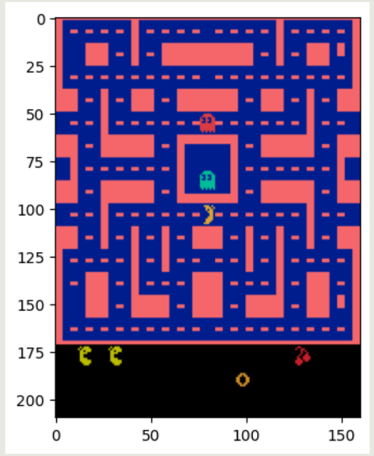
- Frame 2 – Frame 1 (F2 - F1)

Experiment 4: Frame-Diff

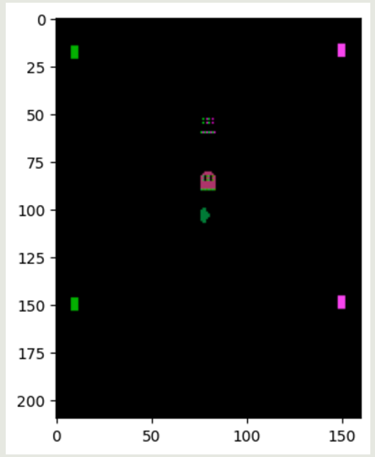
- Sample of a derived frame using Frame-Diff



- Frame 1 (F1)

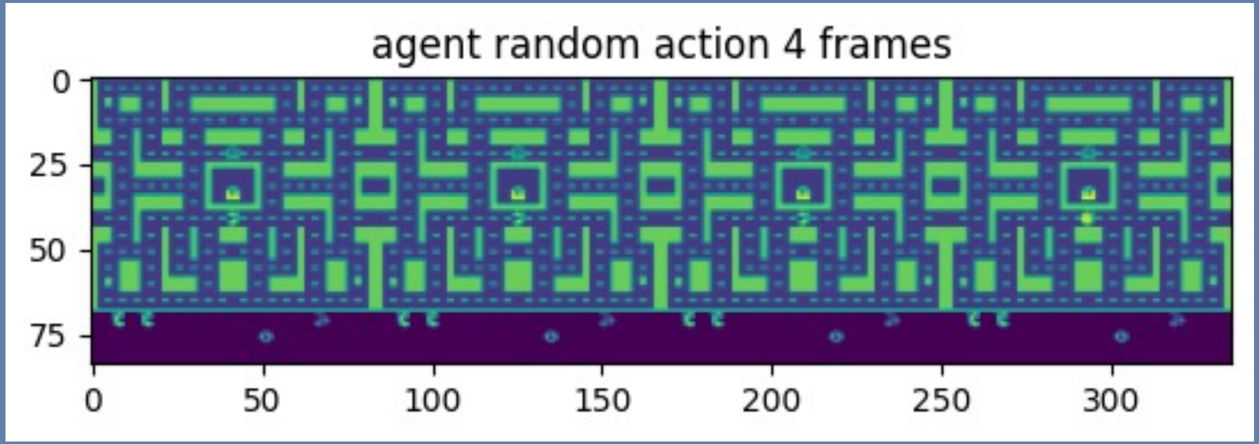


- Frame 2 (F2)

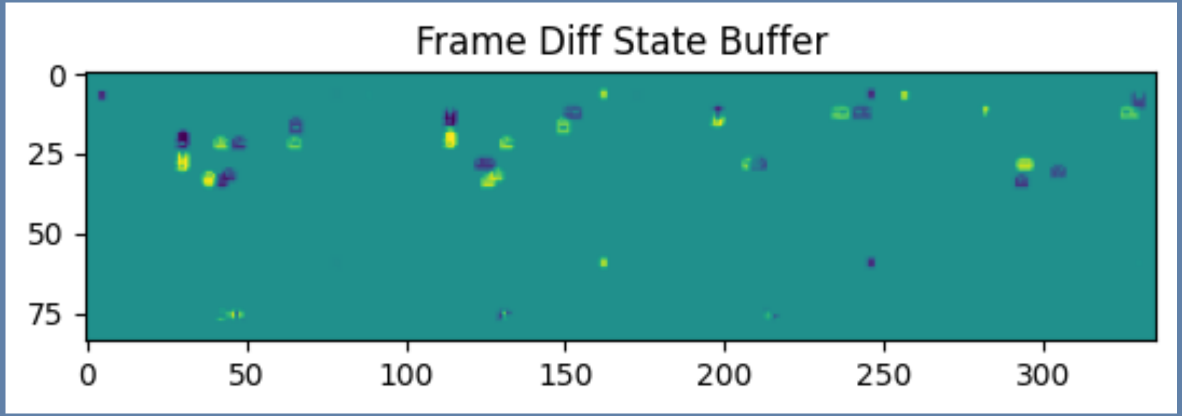


- Frame 2 – Frame 1 (F2 - F1)

Experiment 4: Frame-Diff



- Regular state buffer with 4 frames



- Sample Frame-Diff State Buffer

Experiment 4: Frame-Diff

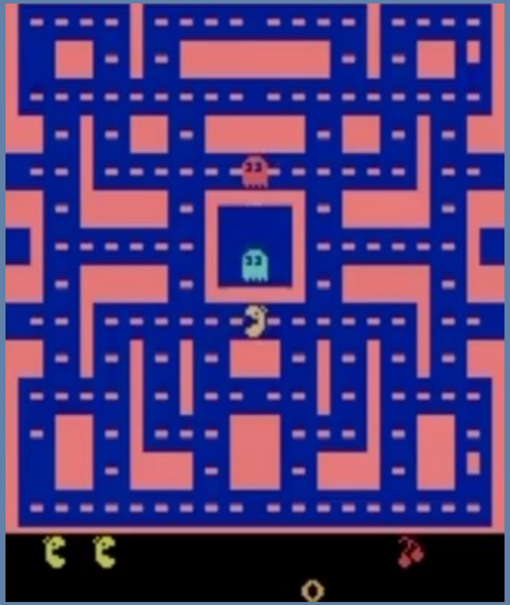
- The Frame-Diff model, at the same learning rate, is able to score 1841 points on average while maintaining a mostly similar standard deviation of 521 among scores.
- 50.53% increase in average agent performance, highly promising since the model was only trained for around 1 hour for 10,000 episodes.
- DQN trained for over 10 million frames, Frame-Diff reaches comparable results with around 1.82 million frames for 50,000 episodes.

	Base model, 10k eps, LR=0.00025	Frame-Diff, 10k eps, LR=0.0001	Frame-Diff, 10k eps, LR=0.00025	Frame-Diff, 30k eps, LR=0.00025	Frame-Diff, 50k eps, LR=0.00025
Trial 1	660	350	2370	1210	2020
Trial 2	2070	820	1960	1410	2400
Trial 3	1070	380	1770	1800	2300
Trial 4	910	570	1710	1400	3730
Trial 5	940	460	1470	1400	2390
Trial 6	1210	480	2540	2000	2560
Trial 7	1360	670	2370	1380	2390
Trial 8	2170	460	1170	1840	2990
Trial 9	1170	460	990	1780	1240
Trial 10	670	570	2060	1100	3260
Average	1223	522	1841	1532	2528
STDEV	522.8352194	140.7756292	521.1834823	300.0666593	684.6377793

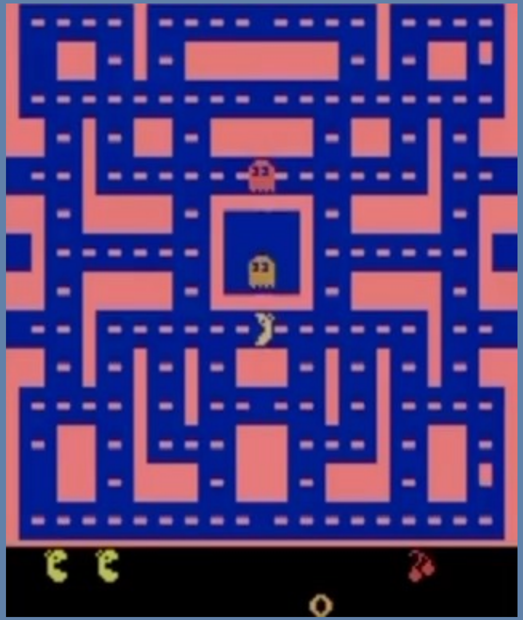
Experiment 4: Frame-Diff

RL Method	Average Score	Source
Frame-Diff	2528	
DQN	2311	Minh et al. [18]
Dueling Architecture DQN (DuDQN)	2250	Wang et al. [21]
Deep Recurrent Q-Network (DRQN)	2048	Hausknecht and Stone [19]
Proximal Policy Optimization Algorithm (PPO)	2096.5	Schulman et al. [20]
DQN with Linear Q-Function	1692	Minh et al. [18]

Frame-Diff Agent Demos



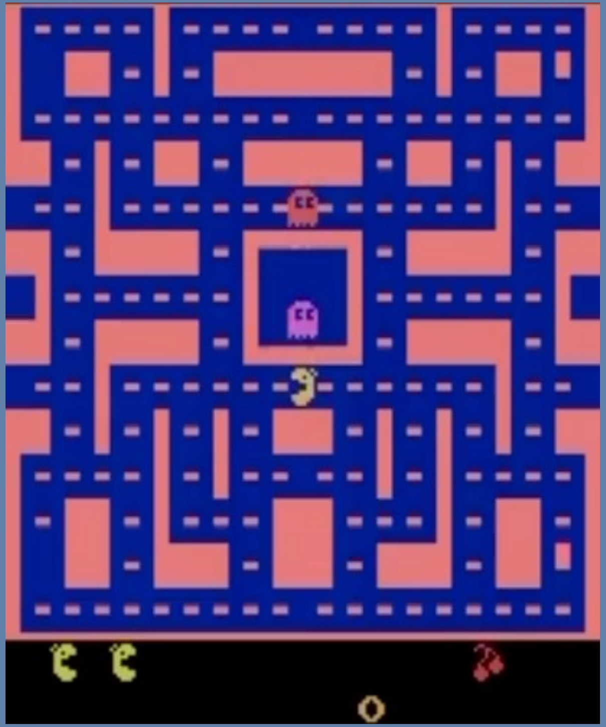
- 10k eps, high score: 2540.



- 30k eps, high score: 2000.

Frame-Diff Agent Demo

- Single highest score achieved at 50,000 episodes: 3730.
- The agent learns the strategy of obtaining invincibility before traversing across the ghost cage (the center of the map), which is the most dangerous area on the map.

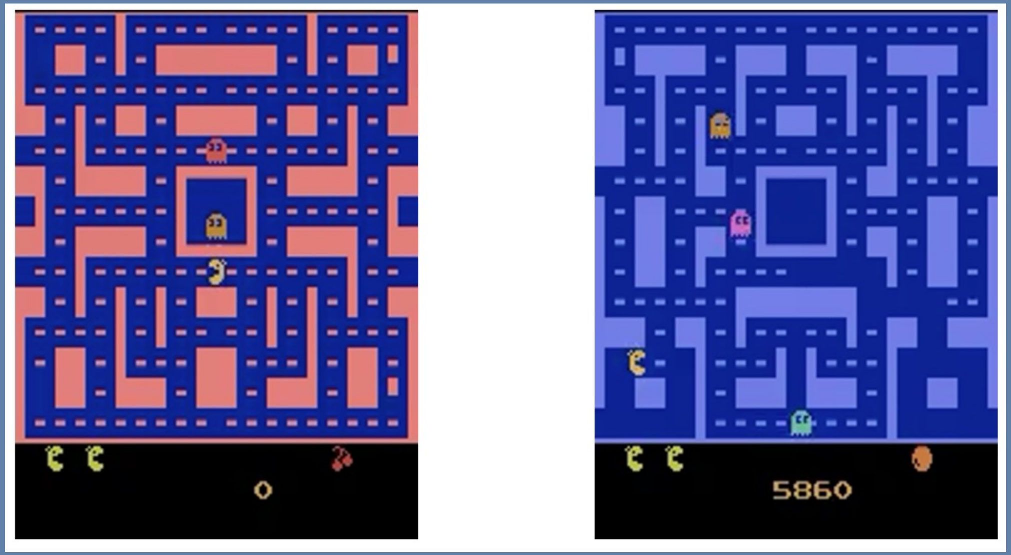


Experiment 5: Transfer Learning

- Transfer learning refers to the problem of using a machine learning model in a different context than what it was trained on.
- As the weights of the CNN are learned as part of the training process, it is possible to “connect” different types of input to the CNN and observe the results.
- Time and resource efficient.
- In Ms. PacMan, an agent could be trained on the first maze of Ms. PacMan and then deployed on subsequent mazes to evaluate its performance on mazes it has not encountered before.
- Maze 2 is an improved version of Maze 1 with faster ghosts that are also better at swarming the agent. Maze 3 is a completely different maze altogether.

Experiment 5: Transfer Learning

Maze 1 and 2 on the left and Maze 3 on the right.



Experiment 5: Transfer Learning

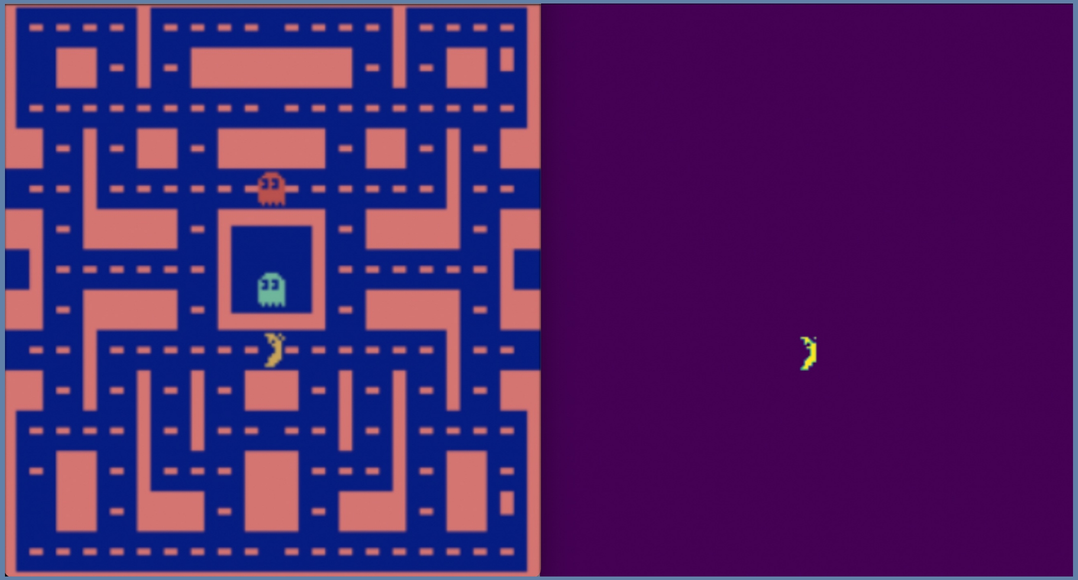
	Maze 1, 10k eps, LR=0.00025	Maze 2, 10k eps, LR=0.00025	Maze 3, 10k eps, LR = 0.00025
Trial 1	660	620	470
Trial 2	2070	720	390
Trial 3	1070	1150	280
Trial 4	910	540	240
Trial 5	940	490	390
Trial 6	1210	710	300
Trial 7	1360	430	200
Trial 8	2170	380	150
Trial 9	1170	470	340
Trial 10	670	610	300
Average	1223	612	306
STDEV	522.8352194	220.5951546	95.93979594

Experiment 6: Transfer Learning with User~Gameplay

- This approach entails isolating and contouring the Ms PacMan character and tracing its movement across the screen.
- Remove all other ghosts, maze walls, food pellets, score and life indicators, etc. via a “yellow mask”, i.e., isolate the yellow color on the screen so that everything disappears except for the Ms. PacMan character.
- Converting the image from Red-Green-Blue (RGB) values to Hue Saturation Values (HSV) is helpful since HSV represents colors in a cylindrical model, which means we can specify ranges easily.

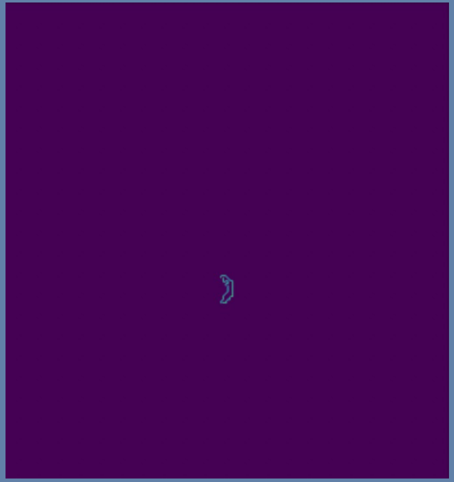
Experiment 6: Transfer Learning with User~Gameplay

- Yellow masking.



Experiment 6: Transfer Learning with User~Gameplay

- After the yellow masking step, the Ms PacMan character is isolated moving across the screen. OpenCV's Canny contouring library provides a handy API for edge detection in images.



Experiment 6: Transfer Learning with User-Gameplay

- Once the contours are obtained, computing the largest contour by area returns the edge for the biggest shape detected on the screen – which, if the yellow mask has been implemented correctly, is Ms. PacMan on the screen.
- To obtain a reward signal, the PyTesseract library is used to convert the score to text.
- The episode completion signal, in the naïve case, is when the frames have all been exhausted.

Experiment 6: Transfer Learning with User-Gameplay

	User-trained, 1 episode	User-trained, 5 episodes	User-trained, 10 episodes	User-trained, 60 episodes
Trial 1	250	210	310	420
Trial 2	340	230	450	300
Trial 3	260	230	250	300
Trial 4	210	230	310	1000
Trial 5	240	230	260	380
Trial 6	240	210	250	360
Trial 7	340	120	250	360
Trial 8	310	230	450	180
Trial 9	340	120	310	180
Trial 10	340	210	310	290
Average	287	202	315	377
STDEV	51.86520992	44.1713834	76.19419634	232.7635901

Experiment 6: Transfer Learning with User-Gameplay

- 1000 points, trained for 60 episodes.
- Agent demonstrates strategy of consuming power-ups and waiting in a corner to eat ghosts.

Experiment 6: Transfer Learning with User~Gameplay

	Base model, 1 episode	Base model, 5 episodes	Base model, 10 episodes	Base model, 60 episodes
Trial 1	130	90	120	320
Trial 2	130	220	120	220
Trial 3	230	160	140	700
Trial 4	130	90	190	210
Trial 5	170	180	110	210
Trial 6	210	70	110	260
Trial 7	170	90	120	210
Trial 8	230	70	190	330
Trial 9	190	90	110	210
Trial 10	170	220	120	190
Average	176	128	133	286
STDEV	38.93013686	60.69962475	31.28720008	153.4202652

Conclusion

- Classic Atari 2600 games provide great environments for the application of reinforcement learning algorithms.
- Our approach in implementing a Deep Q-Agent for one such game, Ms. Pac-Man, has shown promising results across multiple mazes, latency and transfer learning.
- Ultimately, there are a number of thrilling applications of reinforcement learning to solve problems in the real world, not just limited to the use cases that have been described in this project. Our experiments in this project have shown favorable results in adapting DNNs to learn control policies for Ms. PacMan. Further research and innovation in this domain of artificial intelligence only needs a confluence of curious minds and determined effort.

Future Work

- **Guided Training:**
  - There is scope to experiment with augmenting training data to obtain more gameplay videos. One possibility might be to train a generative deep Q network that is trained over far more episodes on a high-performance computing (HPC) cluster.
- **Game Modification:**
  - There are a number of interesting ideas that could be explored if the game itself is modified in some way. For example – if the ghosts were controllable - could dueling Q-agents be trained for the ghosts and the Ms. PacMan character, fighting each other on one game at one time? What if we train the Q-agent on different mazes, maybe even custom mazes, and then try to transfer that to the base PacMan game?
- **Transfer learning from Similar Games:**
  - User gameplay videos of similar games like the original PacMan (1982), or the console version of Ms. PacMan could be used to train an agent for Ms. PacMan.

References

- [1] V. Mnih et al., “Playing Atari with Deep Reinforcement Learning.” arXiv, Dec. 19, 2013 [Online]. Available: http://arxiv.org/abs/1312.5602. [Accessed: May 02, 2023]
- [2] O. Alsing, “Mobile Object Detection using TensorFlow Lite and Transfer Learning,” diva-portal.org, 2018 [Online]. Available: https://www.diva-portal.org/smash/record.jsf?pid=diva2:1242627. [Accessed: May 02, 2023]
- [3] G. Brockman et al., “OpenAI Gym.” arXiv, Jun. 05, 2016 [Online]. Available: http://arxiv.org/abs/1606.01540. [Accessed: May 02, 2023]
- [4] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, “The arcade learning environment: An evaluation platform for general agents,” vol. 47, pp. 253–279, 2013.
- [5] "Part V, Machine Learning, Chapter 22 Reinforcement Learning" in Artificial Intelligence: A Modern Approach. S. Russell and P. Norvig. Fourth Edition, New Jersey: Pearson Education, Inc. 2021, pp. 789-821.

References

- [6] A. P. Badia et al., “Agent57: Outperforming the Atari Human Benchmark.” arXiv, Mar. 30, 2020 [Online]. Available: http://arxiv.org/abs/2003.13350. [Accessed: Apr. 29, 2023]
- [7] "Part V, Machine Learning, Chapter 22 Reinforcement Learning" in Artificial Intelligence: A Modern Approach. S. Russell and P. Norvig. Fourth Edition, New Jersey: Pearson Education, Inc. 2021, pp. 789-821.
- [8] "Mobile Object Detection using Tensorflow Lite and Transfer Learning.". Alsing, Oscar. 2018
- [9] "Deep Learning for Real-Time Atari Game Play using Offline Monte-Carlo Tree Search Planning." Guo, Xiaoxiao, Satinder Singh, Honglak Lee, Richard L. Lewis, and Xiaoshi Wang. Advances in Neural Information Processing Systems 27. 2014
- [10] "Transfer Learning for Related Reinforcement Learning Tasks Via Image-to-Image Translation.". Gamrian, Shani and Yoav Goldberg. PMLR, .2019

References

- [11] "AlphaDDA: Game Artificial Intelligence with Dynamic Difficulty Adjustment using AlphaZero.". Fujita, Kazuhisa. arXiv Preprint arXiv:2111.06266.2021
- [12] "Dynamic Difficulty Adjustment through an Adaptive AI. Silva, Mirna Paula, Victor do Nascimento Silva, and Luiz Chaimowicz.". IEEE. 2015.
- [13] Badia, Adrià Puigdomènech, et al. Agent57: Outperforming the Atari Human Benchmark. arXiv, 30 Mar. 2020. arXiv.org, https://doi.org/10.48550/arXiv.2003.13350.
- [14] K. Simonyan and A. Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition.” arXiv, Apr. 10, 2015 [Online]. Available: http://arxiv.org/abs/1409.1556. [Accessed: Apr. 22, 2023]
- [15] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition.” arXiv, Dec. 10, 2015 [Online]. Available: http://arxiv.org/abs/1512.03385. [Accessed: Apr. 22, 2023]

References

- [16] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, “Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning,” arXiv [cs.CV], Feb. 23, 2016 [Online]. Available: http://arxiv.org/abs/1602.07261
- [17] V. Mnih et al., “Human-level control through deep reinforcement learning,” Nature, vol. 518, no. 7540, pp. 529–533, Feb. 2015.
- [18] M. Hausknecht and P. Stone, “Deep Recurrent Q-Learning for Partially Observable MDPs.” arXiv, Jan. 11, 2017 [Online]. Available: http://arxiv.org/abs/1507.06527. [Accessed: Apr. 29, 2023]
- [19] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal Policy Optimization Algorithms.” arXiv, Aug. 28, 2017 [Online]. Available: http://arxiv.org/abs/1707.06347. [Accessed: Apr. 29, 2023]
- [20] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, “Dueling Network Architectures for Deep Reinforcement Learning.” arXiv, Apr. 05, 2016 [Online]. Available: http://arxiv.org/abs/1511.06581. [Accessed: Apr. 29, 2023]

References

- [21] S. Kapturowski, G. Ostrovski, J. Quan, R. Munos, and W. Dabney, “Recurrent experience replay in distributed reinforcement learning,” International, Jan. 23, 2023 [Online]. Available: https://openreview.net/pdf?id=r1lyTjAqYX. [Accessed: May 02, 2023]
- [22] A. P. Badia et al., “Never Give Up: Learning Directed Exploration Strategies.” arXiv, Feb. 14, 2020 [Online]. Available: http://arxiv.org/abs/2002.06038. [Accessed: May 02, 2023]