Reinforcement Learning in BreakOut Atari Game

Authors

Jirawat Zhou and Ayush Shenvi Pissulakar

Abstract

In this paper, we have attempted to solve the Breakout Atari game with Reinforcement learning, particularly Q-Learning, as the primary approach to solve this problem. We also explored various techniques in training our reinforcement agents. Finally, we examined which techniques best train our agents, such as training with *reflexAgents* vs *randomAgents* and various state-space representations to develop an agent that can perform above the baseline control.

Introduction

We want to train our Breakout agents to intelligently play the Breakout game by attempting to bounce the ball back every time it falls by anticipating the direction and movement of the ball. The motivation of this project is to implement some reinforcement algorithm that can detect some strategies in recovering the optimal policy in hitting the ball back. We will be using the provided *BreakoutNoFrameSkip-v4* in the Arcade Learning Environment (ALE), formerly OpenAI gym, to train our agents.

However, there are several problems that the agents may have encountered in order to retrieve the optimal solution:

1. The environment often provides noisy information when agents observe changes in the environment, which may hinder our reinforcement agent's ability to pick up valuable patterns in recovering optimal strategies.
2. The environment often provides delayed rewards, which means that our agents in the early stages of training may not be able to pick up valuable patterns that can be later exploited to recover optimal policy.
3. Since this project aims to exploit various training techniques and state formulation, we will be using the Q-Learning algorithm, which can be limited compared to Deep-Q Learning or other variants of reinforcement learning algorithms.

Related Works

Several papers have explored the application of reinforcement learning in the Atari game, such as Breakout, Pong, Ms Pacman, Tetris and Flappy Bird. However, the most relevant paper we found to our project was a CS 221 Project Paper by students of Stanford University. Their research paper focused on comparing different RL algorithms, particularly SARSA, SARSA(λ), and Q-Learning and evaluating their performance on Atari Breakout. Unlike our project, their state space was almost entirely created by them using the Pygame library Bricka module and then modified to make it similar to Breakout. They also changed a few rules in how the Breakout game worked compared to the norm and a model-free, off-policy RL approach to help them calculate the performance more efficiently. Unfortunately, after being trained by their Q-learning algorithm, their agent could not provide good results and had around 1.5 points per game. However, their other agents, such as SARSA, Linear Q and Replay Q agents, performed magnificently compared to average human standards (Pierre Berges, Vincent).

Another application of reinforcement learning to arcade games relevant to our project that helped us immensely was a paper released by Google DeepMind in 2015. Their agent was taught to play Atari games based on sensory video input. Their model used convolutional neural networks, which extracted relevant features from the game's video input and heavily relied on their custom feature sets. Their algorithm combines Q-Learning with neural networks and experience replay to decorrelate states and update the action-value function; this was known as Deep Q-Learning. Their algorithm produced terrific results, outperforming humans on nearly 85% of their Breakout games (Human-level Control through Deep Reinforcement Learning).

Our project drew ideas and was inspired by both of these papers by DeepMind and the students from Stanford. However, instead of comparing the performance of each reinforcement learning algorithm, we will instead explore and evaluate the effect of formulating different Markov Decision Processes (MDP). Expressly state representation and reward shaping if there is any positive effect on agent learning performance. The previous papers only choose one formulated MDP formulation, while alternative formulations of MDPs were not explored.

Moreover, looking at these research papers, we were inspired to try different training methods for our algorithm, such as Supervised and Unsupervised training. In this paper, we present an in-depth analysis of the performance of a Q-Learning algorithm made without libraries to create an agent for Breakout using different training methods. We decided to use the RAM method from the OpenAI Gym as it would be more efficient as it gives us access to all of the game variables, with which we created feature sets to run the computation and rewards functions.

Approach

States Space, Rewards Definition

Arcade Learning Environment

The Arcade Learning Environment provided us with many helpful observation types that can be used to train our agents. One of the returned observations is *RAM* observation, where it is represented as 256 bytes in a (128 x 1) array which represents the state of the game in memory. In contrast, grayscale or *RGB* will represent (216 x 216) pixel representation of the state of the game. In this paper, we will mainly focus on *RAM* observation. We believe it would contain sufficient information for our agents to learn about the environment and eventually develop optimal policy.

RAM Annotations

Memory Location	Features
99	Ball x location
101	Ball y location
72	Paddle x location
77	Blocks hit count
103	Ball Velocity*
84	Score
57	Number of lives left

(Mila-Iqia, Atari-representation-learning 2020)

State-Action Representation

We have implemented a total of 2 continuous state representations; however, we quickly realized that the best state representation is the one that encodes just enough information for the Q-Learning agents to pick up optimal strategies but not too much information. Furthermore, this type of state representation will give agents ample search space to search for optimal strategies.

One of the challenges of using *RAM* observation instead of pixel representation is that *RAM* observation may encode noise information which may not be beneficial to be represented in our state-space. Therefore the state space can be reduced by only truncating the bits that would describe the state of the game, which would reduce the state space significantly. Hence the *truncate* Representation of Breakout would be as follows.

$$[(ball_x, ball_y), paddle_x, Direction_{ball}]$$

V1-State

With this *Truncated* Representation, it would reduce the state-action space of the game significant enough for Q-Learning to converge on suboptimal/optimal policy. However, although it may significantly reduce the state-action space, it still encodes some duplicate or redundant information that can be represented compactly.

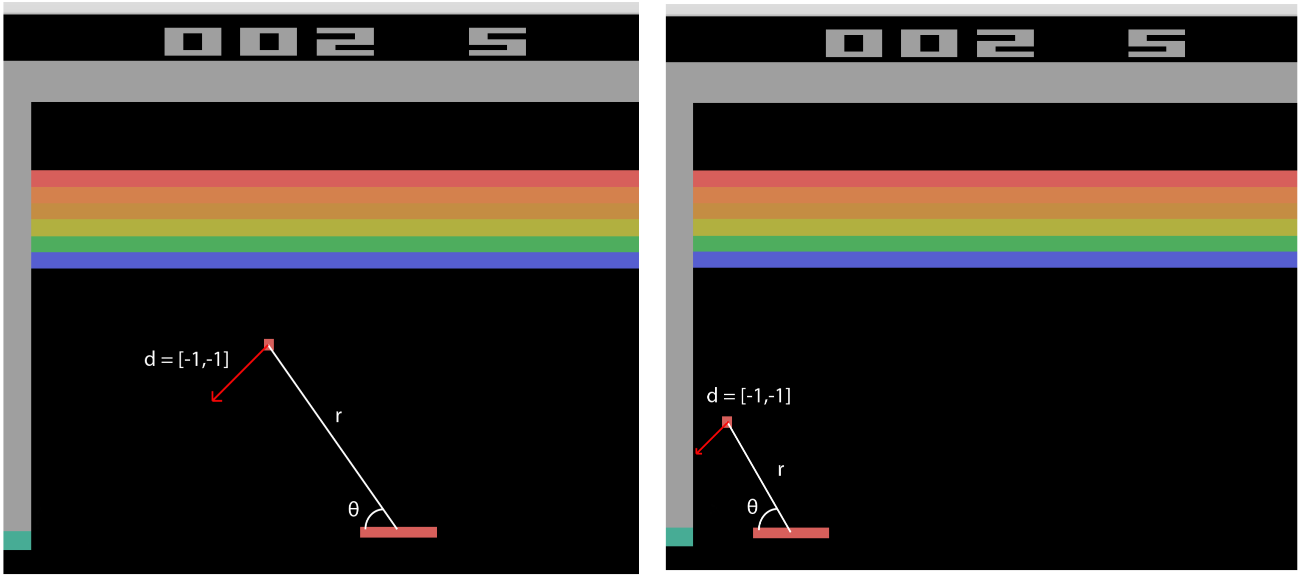
Hence *V2* will mainly focus on reducing the information from *V1,* whereas V2 will use relativity between the ball and paddle as the primary technique to optimize state representation.

$$[(r, \theta), Direction_{ball}, WallSensor]$$

V2-State

Where the *WallSensor* is used to detect if the ball is going to bounce off the wall to differentiate between the following two states:

Figure 1: Wall Sensor State Representation



When given these two states, the optimal action taken by the agents should be different. For instance, in the first instance, the agents should have taken action *left,* while in the second instance, the agents should have taken action to stay or *right* as the ball may bounce off the surface in the opposite direction. Hence the WallSensor will be turned off (i.e. false) in the first instance and turned on (i.e. True) in the second instance.

Rewards

As agents play the game, the default ALE rewards function will be given +1 reward to an agent only if a block is broken and 0 otherwise. The issue with this reward function is that there are significant temporal gaps between the state where the agents receive the rewards and the state where the action takes place to return the ball. Furthermore, there is no penalty when an agent takes the wrong direction (i.e. moving away from the ball instead of moving closer).

In order for our agent to learn quickly, we have provided a reward of +1 when the agent can hit the ball back and a reward of -1 when the agent fails to hit the ball back. Which can be formally defined as follows

$$R(s, a, s') = \left\{ \begin{matrix} 1, & \text{when a paddle bounce the ball back} \\ -1, & \text{when agent lose life} \\ 0, & \text{Otherwise} \end{matrix} \right\}$$

Reward Function

Methods

We decided to use a model-free reinforcement algorithm, specifically the Q-Learning algorithm and *SARSA*, rather than model-based learning. As a result, the breakout agent does not have access to the Transition function defined in Markov Decision Processes (MDP).

Q-Learning is a form of model-free, **off-policy** learning. The agents estimated the utilities of a given state-action pair by performing actions on particular states and observed its reward to update the utilities accordingly. Where at each time step, the following action is given by acting policy where given a state *s* the policy will select action *a* that maximizes the Q(s', a') value. Which can be formally defined as the following bellman equation

$$Q(s, a) = (1 - \alpha)Q(s, a) + \alpha(R(s, a, s') + \gamma\, max_{a'} Q(s', a'))$$

Q-Learning Bellman Equation

SARSA is also a form of model-free, **on-policy** learning. Like Q-Learning, the agents estimated the utilities of a given state-action pair by performing action on particular states and updating the value estimated according to the observed rewards. However, the main difference between SARSA and Q-Learning agents is that at each time step, the following action is selected based on current policy instead of epsilon-greedy that maximizes Q(s', a'). Which can be formally defined as the following bellman equation:

$$Q(s, a) = (1 - \alpha)Q(s, a) + \alpha(R(s, a, s') + \gamma Q(s', a'))$$

SARSA Bellman Equation

Therefore generally*, SARSA* will converge on the optimal policy under the assumption that the current policy will be used to generate the agent's experience. In contrast, Q-Learning will converge on the optimal policy after generating enough experience and switching over epsilon-greedy policy without relying upon the current policy. Hence, we have selected Q-Learning as the primary approach to this problem. We want our agent to rely upon experience to generate optimal policy rather than relying on the current policy that may not have been optimal/suboptimal.

Training

We have trained our agents on two models to ensure that our Q-Learning agents converge on some optimal policy; one of the models is a *reflexAgents* where agents will always move in the direction closest to the ball. The other model is *randomAgents,* where the agents will randomly select action to be acted on the environment at a given timestep.

Although we found that agents trained on the *reflexModel,* the initial learning performance performed significantly better than agents trained on *randomModel,* however, at a certain point in training, the agent that trained on *reflexModel* failed to exhibit a significant increase in learning performance after ~100,000 episodes.

To counter this low learning performance, the agents trained on *reflexAgents* will be switched to *randomAgents* after no significant increase in learning performance to promote more exploration instead of exploitation of the current *ReflexAgents* model.

PseudoCode for Reflex Agents

```
Algorithm 1: ReflexAgents
Input: state
Output: action
$ball_x, ball_y, paddle_x, direction = state$    // Extract information from the
  gameState
if $ball_x > paddle_x$ then
    /* Ball is relatively located at the Right of the paddle            */
    return Right
else if $ball_x < paddle_x$ then
    /* Ball is relatively located at the Left of the paddle             */
    return Left
else
    return Stay
```

The above is the ReflexAgents model formally defined using *Truncated State Representation* with specific (x,y) coordinate of the ball and x coordinate of the paddle.

PseudoCode For Epsilon Greedy Action Selection For **Supervised** Training

```
Algorithm 2: getAction
  Input: state
  Output: action
  if $flipCoin(\epsilon)$ then
      /* Explore by invoking ReflexAgents Model to get action                */
      return $ReflexAgents(state)$
  else
      /* Exploit by taking next step lookahead to get the next action that
         maximizes utilities                                                  */
      return $argmax_{action}Q(state, action)$
```

PseudoCode For Epsilon Greedy Action Selection For **Unsupervised** Training

```
Algorithm 3: getAction
  Input: state
  Output: action
  if $flipCoin(\epsilon)$ then
      /* Explore by randomly selecting action                                 */
      return $random.choice([Left, Right, Stay])$
  else
      /* Exploit by taking next step lookahead to get the next action that
         maximizes utilities                                                  */
      return $argmax_{action}Q(state, action)$
```

Results & Evaluation

The average points per episode across multiple episodes compare different approaches to evaluate our agents' performance. As illustrated in Figure 1, *The Average Points Per Episodes Over 100 Episodes* for each different approach in comparison.

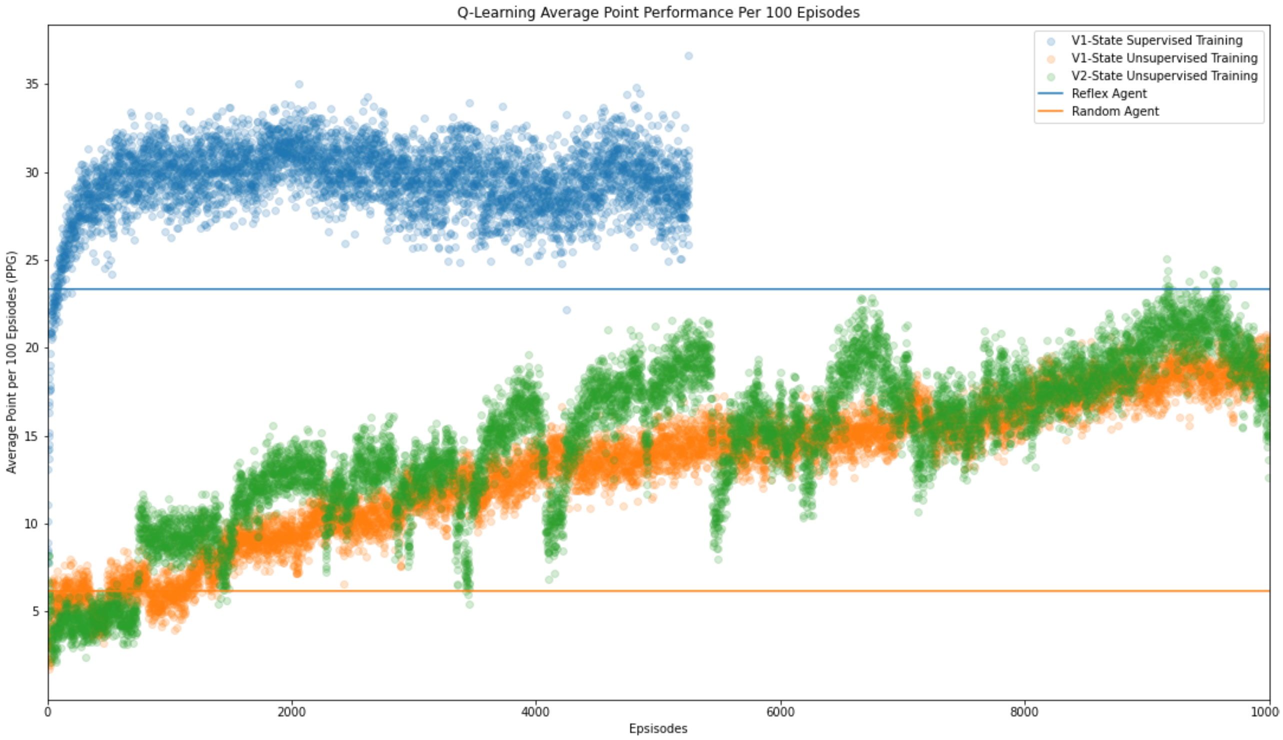


Figure 1: Average Points Per Episodes across 100 episodes for 1,000,000 episodes of gameplay (~ 48 hours for each model). Where epsilon greedy ϵ is at 0.02 and learning rate α is at 0.01 with a discounted rate γ of 0.99. Where V1 Supervised Training represents the agent that trained with *Truncated Representation* and ReflexAgents, while V1 Unsupervised will be trained with RandomAgents with the same state representation. Lastly, V2 Unsupervised represents the agent that trained with *Polar Representation* and RandomAgents.

All formulated agents have shown an increase in average performance, which can be seen in *Figure 1*. While the learning rate varies between agents to agents, V1 supervised agents perform the best among other agents during the initial training stages. However, the learning rate of V1 and V2 unsupervised agents continue to increase at a sustainable rate with more training. In contrast, the learning performance of V1 supervised agents seem to drop/plateau after converging on some particular policy.

The tabulated result can be seen in Table 1. The average score for *Unsupervised V1 Representation* is 18.59 points during the 900,000 to 1,000,000 episodes, and *Unsupervised V2 Representation is* at 20.35 points during the same training episodes. While the *Supervised V1 Representation* seems to converge at ~29 points during the 100,000 - 500,000 episodes. One interesting observation can also be drawn from the tabular result is that the agents with *Polar Representation* consistently perform better than *the Truncated Representation,* even though the *Polar Representation* encapsulates less information about the states of the game when compared to *Truncated Representation,* which reaffirms our intuition that formulation of states space has a direct impact on learning performance of Q-Learning agents. The optimal state definition should describe all information necessary for agents to navigate the world, while additional or redundant features may otherwise inhibit the agent's training performance.

Nonetheless, the performance difference between the *Truncated Representation* and *Polar Representation* is not significant to be seen in *Figure 1*, and more training may be needed to exacerbate the difference.

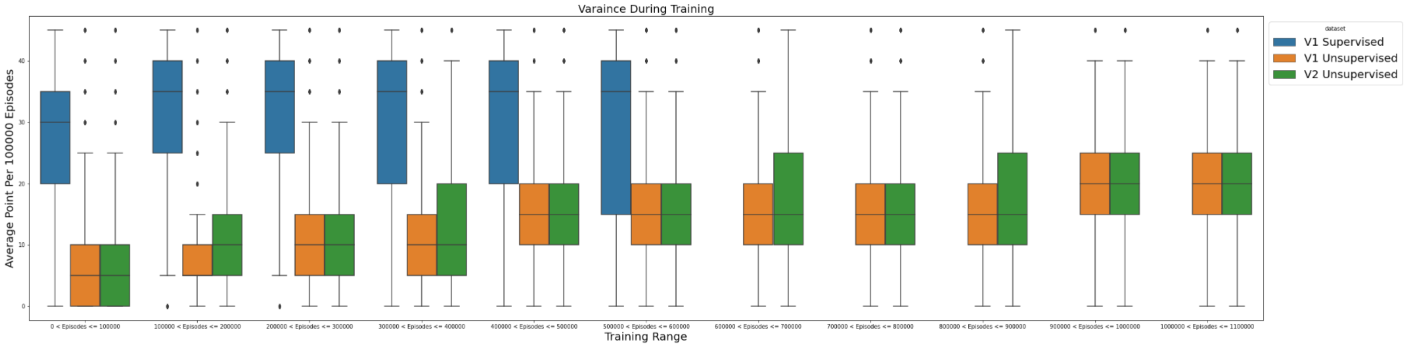
Table 1: Tabulated Result of Average Points Per 100,000 Episodes

Training Episodes	Model	Performance (PPG)
0 < Episodes <= 100000	V1 Supervised	28.05
0 < Episodes <= 100000	V1 Unsupervised	5.82
0 < Episodes <= 100000	V2 Unsupervised	5.77
100000 < Episodes <= 200000	V1 Supervised	30.39
100000 < Episodes <= 200000	V1 Unsupervised	8.16
100000 < Episodes <= 200000	V2 Unsupervised	10.57
1000000 < Episodes <= 1100000	V1 Unsupervised	19.04
1000000 < Episodes <= 1100000	V2 Unsupervised	19.98
200000 < Episodes <= 300000	V1 Supervised	30.34
200000 < Episodes <= 300000	V1 Unsupervised	10.30
200000 < Episodes <= 300000	V2 Unsupervised	12.47
300000 < Episodes <= 400000	V1 Supervised	29.33
300000 < Episodes <= 400000	V1 Unsupervised	12.21
300000 < Episodes <= 400000	V2 Unsupervised	13.79
400000 < Episodes <= 500000	V1 Supervised	29.43
400000 < Episodes <= 500000	V1 Unsupervised	13.60
400000 < Episodes <= 500000	V2 Unsupervised	15.70
500000 < Episodes <= 600000	V1 Supervised	28.83

500000 < Episodes <= 600000	V1 Unsupervised	14.56
500000 < Episodes <= 600000	V2 Unsupervised	16.51
600000 < Episodes <= 700000	V1 Unsupervised	15.06
600000 < Episodes <= 700000	V2 Unsupervised	17.12
700000 < Episodes <= 800000	V1 Unsupervised	15.90
700000 < Episodes <= 800000	V2 Unsupervised	16.31
800000 < Episodes <= 900000	V1 Unsupervised	17.69
800000 < Episodes <= 900000	V2 Unsupervised	18.35
900000 < Episodes <= 1000000	V1 Unsupervised	18.59
900000 < Episodes <= 1000000	V2 Unsupervised	20.35

While the variance of each agent during training can be seen in Figure 2, where more significant variance can be observed with *V1 Supervised* agents, where the variance continues to get more significant as it progresses through the training. One of the reasons why such a significant variance might be the interplay between ReflexAgents and RandomAgents. As the agents converge to a sub-optimal policy during training with ReflexAgents, the agents are switched to RandomAgents to promote exploration of the environment instead of exploiting the ReflexAgent's behavior.

Figure 2: Performance Variance Throughout Training



While exploration is encouraged throughout V1 supervised training, however, it did not yield an increase in the average performance of the agent instead, the agent keeps exploring new optimal/suboptimal policies that may differ from the policy that agents have learned through *ReflexAgent*, which may explain a vast and increasing variance during the 300,000 - 600,000 episodes. Another possible explanation is the interplay between different epsilon values (0.5 from 0 - 100,000 and 0.2 from 100,000 - 500,000) throughout the training; however, this may not explain the increase in the variance of average performance over time, but it may be a contributing factor that may explain more significant variance when compared to V1 and V2 unsupervised training.

Finalized Policy

We also explored the Policy or Q(s, a) value of the agent to understand how the agent learned how to play the game. We will be using the **unsupervised** with ***Truncated Representation*** *(V1)* agent. It is easier to interpret and show the $argmax_{a} Q(s, a)$ to inferes the policy generated at the end of the Training after ~1,000,000 episodes.

Table 2: Sample Q(s,a) Value Table of V1 Unsupervised Model compared with ReflexAgent

Truncated Representation $[(ball_{x}, ball_{y}), paddle_{x}, Direction_{ball}]$	**Action** $[Stay, Right, Left]$	**Q-Values**	$\pi_{ReflexAgents}$
(((65, 113), 169, (1, 255))	Stay	0.06772679	**Left**
(((65, 113), 169, (1, 255))	Right	0.05471271	
(((65, 113), 169, (1, 255))	**Left**	**0.25847482**	
(((67, 115), 169, (1, 255))	**Stay**	**0.26325462**	**Left**
(((67, 115), 169, (1, 255))	Right	0.04955189	
(((67, 115), 169, (1, 255))	Left	0.02603028	
(((173, 117), 152, (1, 254))	Stay	1.07E-16	**Right**
(((173, 117), 152, (1, 254))	**Right**	**6.77E-09**	
(((173, 117), 152, (1, 254))	Left	3.54E-15	
(((200, 135), 148, (1, 254))	Stay	2.42E-12	**Right**
(((200, 135), 148, (1, 254))	**Right**	**3.13E-06**	
(((200, 135), 148, (1, 254))	Left	1.42E-24	

From the above-Tabulated Q(s, a) values for *V1 Unsupervised Agent,* it can be shown that at 1,000,000 episodes, the agents converge on a particular policy that almost closely mimics the *ReflexAgents.* If the ball is relative to the left side of the paddle, then the *Left* action will be taken, while if the ball is relative to the right side of the paddle, then the *Right* action will be taken.

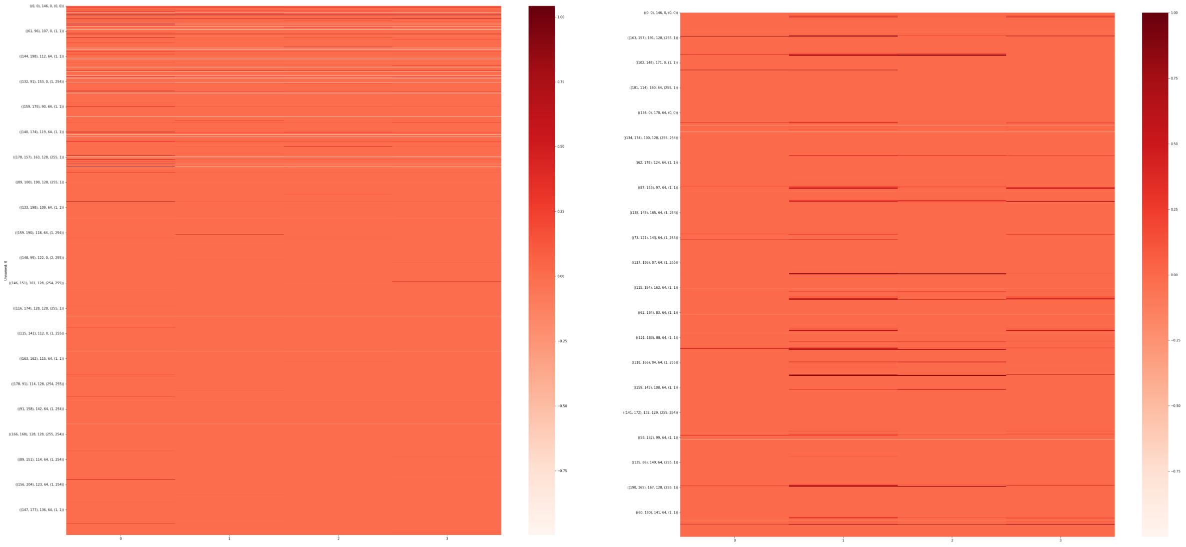
Discussion

From all of the results shown above, it can be seen that different state representation directly impacts the agent learning performance, as smaller state space may help agents explore various states at a faster rate. However, suppose the state formulation is too vague or does not contain enough information about the state of the world. In that case, the learning performance may fluctuate, as illustrated in the Polar representation in Figure 1. Higher variance during training can be illustrated in Figure 2. While more training may be needed to determine if the agent that trained with Polar Representation may converge upon the same policy as the agent that trained with Truncated Representation, from initial observation in Figure 1, we can see that agent with Truncated Representation learning performance start to decrease at 500,000 - 600,000 intervals, which may imply that when the Q-Learning ultimately converges on the policy generated from the agent with Polar Representation, it may instead be a suboptimal policy rather than an optimal one when compared with a policy that generated with the agent with Truncated Representation.

One of the reasons why *Polar Representation* learning performance tends to fluctuate around might be due to its state definition containing relative position rather than complete observation of the environment, where one particular state may be common with other particular states, for example, when the state that describes the environment where the ball is moving to the left and the paddle position is at 200 is similar to the state where the actual environment is that the ball is moving to the left, but the paddle is at 0. Thus the optimal action in the first instance should be moving to the *left,* while in the second instance, the optimal action may be instead *Stay* or move *Right*. However, we have countered this by introducing the *WallSensor* component to the state representation. However, it still may not suffice. Hence this may largely contribute to fluctuating performance, and when the Q-Learning eventually converges, it may be converging on a sub-optimal policy rather than an optimal one.

Meanwhile, when compared with the agent that trained with *Truncated Representation,* if the state representation contains enough information about the state of the world, such as the MDP formulation described by *Truncated Representation, such as* (V1 Unsupervised agent) tends to have a consistent increase or non-fluctuation in learning performance and lower variance during training. The *Truncated Representation* did not use all of the given *RAM* observation, but it contained just enough information for Q-Learning agents to represent the state of the environment.

Figure 3: Q(s,a) Heatmap of *Truncated Representation* for Unsupervised and Supervised



*This Heatmap represents the Q(s,a) value of an Agent with Truncated Representation that trained with **RandomAgents**.*

*This Heatmap represent the Q(s,a) value of an Agent with Truncated Representation that trained with **ReflexAgent** before interplay with RandomAgent*

Where darker red represents positive value associated with state, and lighter red represents negative value associated with the state, and neutral orange represents unexplored state.

Another way that we have come up with to increase the learning performance of the agent is to train the agent with some sub-optimal policy that we have defined as *ReflexAgents,* however from the above result it show a promising increase in learning performance, however the variance of the performance is also tend to be higher when compared with agent that trained against *RandomAgents.* Although we already discussed it through inspecting *Figure 1 and Figure 2,* but the effect can also be seen in the generated policy. When we have examined the policy of *Supervised Agents* and *Unsupervised Agents* we have noticed a difference in policy, where the policy that is generated by the *Supervised Agent* tend to have higher Q-Value on state-action pairs that reflex the *ReflexAgents,* however for state-action pair that does not reflex the *ReflexAgent* is rarely updated with non-zero Q-value. From this observation it implies that the *Supervised Agent* tends to exploit the rewards, but rarely explore other possible actions that may get to the rewards, for instance predicting where the ball will land instead of following the ball. Which can be seen in *Figure 3* above, where there are concentrated spots of dark red. Hence this leads to the majority of the state space not being explored but rather explored on only fews particular states-action pairs. This may possibly lead to a repetitive behavior that mimics the *reflexAgent* rather than generating optimal policy. Therefore after interplay with the *RandomAgents* the performance actually drops as it explores the unexplored states more often that may lead to changes in optimal value for some states.

While, for policy generated by *Unsupervised agents* there are significantly higher differences between state-action pairs that have positive value and negative value, which implies that the agents have learned to exploit actions that maximize rewards, but also minimize punishment (i.e predicting where the ball will fall). This observation in policy also reflects upon Figure *2* and *Table 2,* where the variance is consistently lower for agents that trained with *RandomAgent,* Although we already have previously discussed that the interplay between different epsilon values may contribute to larger variance for *Supervised Agent*. However after examining the policies we believe that policies generated from *Supervised* and *Unsupervised* agents are fundamentally different and that may contribute to different learning performance.

Although we believe that if the *Supervised Agent* continues to be trained on the *RandomAgent* the variance will reduce and eventually converge on optimal policy.

Conclusion

In conclusion, we have implemented various definitions of MDP, such as different state definitions and reward shaping. We also attempted to train our agent with a sub-optimal policy to examine different learning performances between agents trained with *ReflexAgents* and *RandomAgents.* In the end, we show that different state definitions directly impact learning performance. From our data, a smaller state definition *(Polar Representation)* leads to an increase in learning performance faster. However, it does not always benefit the agent's learning as a smaller representation may contain less information about the states of the world, which may eventually converge on a sub-optimal policy rather than an optimal policy.

Nonetheless, more training is needed to see if the agent with *Polar Representation* performance diverges off from the agent with *Truncated Representation.* While the agent trained with the *ReflexAgent* showed a promising initial learning performance, it failed to converge on the optimal policy as the agents began to interplay with *randomAgents. However*, more training may be needed for all agents to converge on an optimal policy eventually.

To continue with our future work, we may implement those formulated definitions of MDP on other types of model-free reinforcement algorithms such as SARSA, PPO, PG and vice versa to explore the difference in MDP formulation on the learning performance of different reinforcement learning algorithms.

Acknowledgement

Jirawat Zhou: Formulated MDP and Implemented Q-Learning Agents and Report
Ayush Shenvi Pissulakar: Supervised Training of agents, Data Collection and Report

User Graphics and Training Environment Provided by Arcade Learning Environment.
Bellemare, M.~G. and Naddaf, Y. and Veness, J. and Bowling, M., The Arcade Learning Environment: An Evaluation Platform for General Agents, Journal of Artificial Intelligence Research

References

1) ”Human-level Control through Deep Reinforcement Learning.” DeepMind. N.p., n.d. Web. 12 Dec. 2016.
2) Karpathy, Andrej, “Deep Reinforcement Learning: Pong From Pixels”. karpathy.github.io. N.p., 2016. Web. 7 Dec. 2016.
3) Sutton, Richard S., and Andrew G. Barto. “Reinforcement Learning: An Introduction”. Cambridge, MA: MIT, 1998. Print.
4) Developer, Author: Satwik Kansal Software, et al. “Reinforcement Q-Learning from Scratch in Python with Openai Gym.” *Learn Data Science - Tutorials, Books, Courses, and More*, https://www.learndatasci.com/tutorials/reinforcement-q-learning-scratch-python-openai-gym/.
5) Rana, Ashish. “Introduction: Reinforcement Learning with Openai Gym.” *Medium*, Towards Data Science, 13 July 2021, https://towardsdatascience.com/reinforcement-learning-with-openai-d445c2c687d2.
6) Kathuria, Ayoosh. “Getting Started with Openai Gym.” *Paperspace Blog*, Paperspace Blog, 9 Apr. 2021, https://blog.paperspace.com/getting-started-with-openai-gym/.
7) Pierre Berges, Vincent, et al. *Reinforcement Learning for Atari Breakout*.
8) Defazio, Aaron, and Thore Graepel. “A comparison of learning algorithms on the arcade learning environment.” arXiv preprint arXiv:1410.8620 (2014)

Appendix

Repository: https://github.com/Jirawatz/BreakoutAI-V5