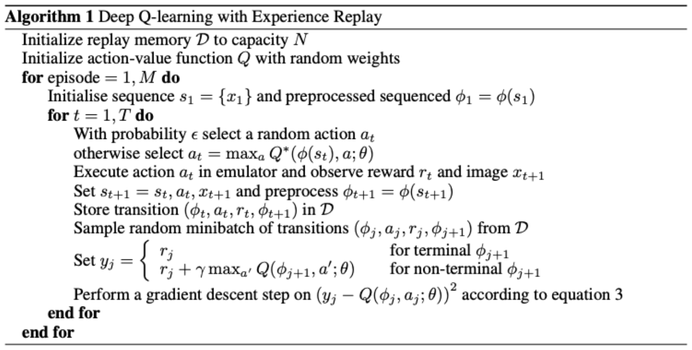
# ECSE 526
# Lab Report 3
# Reinforcement Learning for Ms. Pac-Man with Python and Ale-py using Experience Replay

Yajiv Luckheenarain (260986423)

**This project employs Python and Ale-py to train a Ms. Pac-Man agent through Experience Replay learning. Leveraging Ale-py's interface with the Atari Learning Environment, our agent refines strategies over iterations by updating its value function based on observed outcomes. The Python environment, utilizing libraries like NumPy and TensorFlow, facilitates efficient neural network implementation for decision-making. Preliminary results showcase the agent's adept navigation, evasion of ghosts, and pellet collection. This work contributes to reinforcement learning, illustrating the practical use of Python and ale-py in training agents for complex gaming environments.**

## I. Introduction

IN this project, the ale-py interface streamlined the loading of the Ms. Pac-Man binary file, providing access to essential game elements for constructing a comprehensive state space. The state representation, vital for agent learning, was crafted using an experience replay model with 12 input features. These include Ms. Pac-Man's coordinates, Euclidean distances to each ghost, ghost states where '1' represents an enemy ghost and '0' represents a vulnerable ghost, the player's remaining lives and the total score achieved in the game. The outputs of the model are the four possible moves Ms. Pac-Man could execute (up, down, left, right). To balance exploration and exploitation during training, an epsilon-decreasing learning strategy was implemented. Real-time visualization of the evolving game state, facilitated by the Pygame library, offered insights into the learning process. This project contributes to reinforcement learning, demonstrating the practical application of these techniques in training agents for complex gaming environments like Ms. Pac-Man. The following figure shows the Experience Replay pseudo-code implemented for this agent. [1]



```
Algorithm 1 Deep Q-learning with Experience Replay
  Initialize replay memory $\mathcal{D}$ to capacity $N$
  Initialize action-value function $Q$ with random weights
  for episode $= 1, M$ do
      Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
      for $t = 1, T$ do
          With probability $\epsilon$ select a random action $a_t$
          otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
          Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
          Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
          Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
          Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
          Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
          Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
      end for
  end for
```

Fig. 1: Experience Replay Pseudo-Code

## II. Description of Approach to Generalization

### A. *Distance Metric Between Two States*

The distance metric between two states for this implementation of an RL agent is denoted as

$$D(S_1, S_2) = \sqrt{\sum_{i=1}^{12} (S_{1i} - S_{2i})^2}$$

where each state $S_i$ is characterized by a 12-element feature vector. In this context, $S_1$ and $S_2$ denote two distinct states, and $S_{1i}$ and $S_{2i}$ represent the $i$-th components of the respective 12-dimensional feature vectors. The Euclidean distance calculation is a fundamental measure of dissimilarity between these states, considering the differences across all 12 dimensions of the feature vectors. This formula encapsulates the spatial relationships and attributes embedded in the feature vectors, providing a quantitative metric for the RL agent to gauge the dissimilarity or proximity between different states in the Ms. Pac-Man game environment.

### B. *Rationale for the choice of components in the distance metric*

The state representation employed by the RL agent comprises 12 key features, including the coordinates of Ms. Pac-Man, the Euclidean distances between Ms. Pac-Man and each ghost, the states of the ghosts (where '1' denotes a bad, non-vulnerable ghost, and '0' signifies a good, vulnerable ghost), the total score achieved in the game, and the remaining lives of the player. The Euclidean distances between Ms. Pac-Man and the ghosts serve as a pivotal component of the distance metric. The rationale behind this choice is quite straightforward. The agent is encouraged to maximize the Euclidean distances when ghosts are in a non-vulnerable state, promoting evasion and strategic positioning. Conversely, when ghosts enter a vulnerable state, the model learns to minimize these distances allowing for opportunities to maximize their game scores as 'eating' vulnerable ghosts results in a positive reward score. This dynamic approach to distance metrics enhances the agent's adaptability to varying game scenarios, optimizing its decision-making process in response to the changing states of the ghosts.

## III. Results of Generalization

### A. Different approach to generalization

The 12 features chosen to represent the state of the game space to the agent were described earlier. Two different iterations of the chosen features were also chosen to train the model. The first iteration was to use 20 input features where the 8 additional features represent the raw position (x,y) of each of the four ghosts in the Ms.Pacman game. Although these features might be useful in training the model to determine complex spatial relationships between the player and the ghosts, the computational resources required to train these differing models can be expensive. A third iteration of input features to the model was to add CNN layers to my model and feed the RGB values of the game's pixel values as input features. Once again, training with those features was too computationally expensive. The following graph shows the agent's performance on these two different types of configurations.

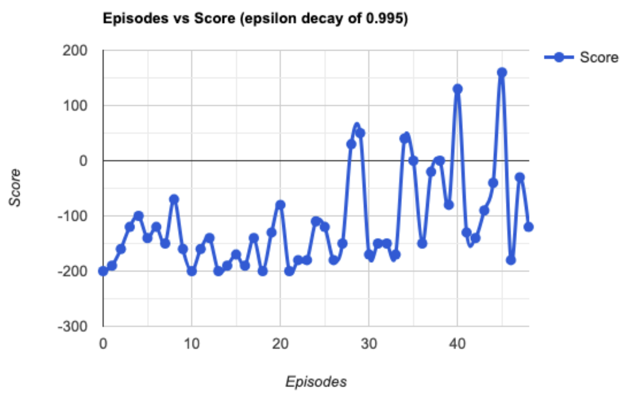


Fig. 2: Score vs Episode for 12 input features and epsilon decay of 0.995

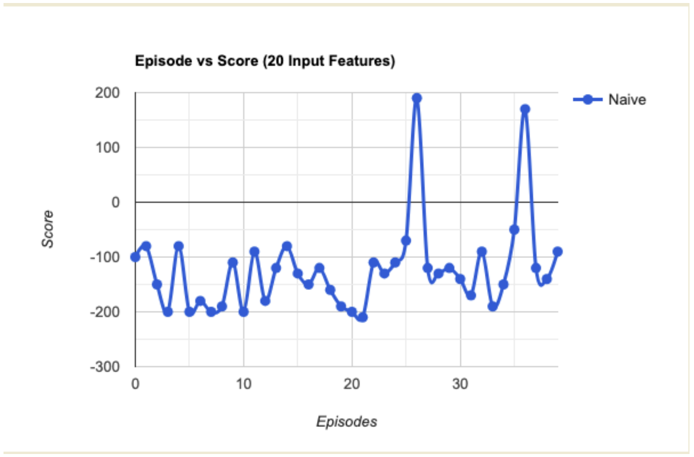


Fig. 3: Score vs Episode for 20 input features

### B. Consequences of approaches to agent's Behaviour

From Fig.**??**, it can be clear that the agent is slowly learning over time as its score has the trend to increase over the episodes. However, it is not linearly growing as there is a lot of fluctuation where the agent's achieved score ”dips”. Nevertheless, the trend shows that the agent is effectively learning over episodes.

From Fig.**??**, we can see that the agent achieves scores that fluctuate, but remain relatively low, with some random ”spikes” in performance.

It is obvious that between Fig.**??** and Fig.**??**, the agent with the least amount of features performs better in the early stages of training. That is likely due to forming relationships between features is easier to achieve initially with a lower amount of features for the model to consider.

## IV. Approach to exploration

### A. Expression for optimistic prior

Incorporating an optimistic prior for state-action pairs, the RL agent employs the Greedy in the Limit with Infinite Exploration (GLIE) strategy. It balances exploration and exploitation using an epsilon ($\epsilon$) parameter. Initially set to 1, indicating 100 percent random actions for thorough exploration:

$$\epsilon_0 = 1$$

As the agent learns, epsilon undergoes exponential decay:

$$\epsilon_t = \epsilon_0 \cdot e^{-\alpha t}$$

Here, $\epsilon_t$ is the exploration parameter at time step $t$, $\epsilon_0$ is the initial value, $\alpha$ is the decay rate, and $t$ is the current time step. This dynamic adjustment allows the agent to shift from exploration to exploitation as it learns, adapting to the environment.

### B. Agent's Behaviour Guided by Exploration-Exploitation Strategy

Throughout the agent's training, it employs a dynamic Exploration-Exploitation strategy to decide its actions. This strategy involves a balance between two key components: policy-driven exploitation and random exploration. The agent's policy guides action selection based on learned knowledge, reflecting exploitation. Concurrently, the exploration component introduces randomness, allowing the agent to occasionally choose random actions based on the epsilon term. Initially set to 1, this epsilon term signified that 100 percent of actions taken by the agent are random. As $\epsilon$ undergoes decay, the agent gradually reduces its random exploration.

### C. Rationale Behind Exploration-Exploitation Strategy

This adaptive strategy ensures a flexible approach, allowing the agent to exploit learned policies while maintaining a capacity for exploration, crucial for discovering optimal actions in complex environments.

## V. Results of Exploration

### A. Epsilon Decay of 0.995

Fig.**??** already showed the behaviour of an agent trained using GLIE (Greedy in the Limit with Infinite Exploration) where the epsilon decay is set to 0.995.

### B. *Epsilon Decay of 0.7*

The following figure shows the score achieved by the agent when employing an epsilon decay of 0.7.

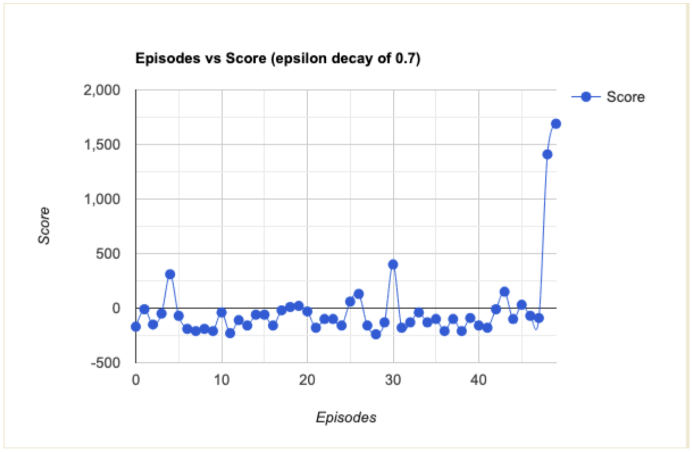


Fig. 4: Score vs Episode for 12 input features and epsilon decay of 0.7

### C. *Analysis on Exploration Function*

Two agents were trained on the Ms. Pac-Man game using the GLIE (Greedy in the Limit with Infinite Exploration) algorithm. The first agent utilized an epsilon decay rate of 0.995, while the second agent employed a lower decay rate of 0.7. Throughout the training process, the first agent exhibited a gradual improvement in performance, steadily learning and adapting to the game environment as can be seen in Fig.**??**. In contrast, the second agent displayed relatively low overall performance, with occasional random peaks in its scores as can be seen in Fig.**??**. Interestingly, despite its seemingly inconsistent progress, the second agent ultimately achieved the highest overall score.

From the results previously shown, in the case of RL agents trained on Ms.Pacman, it suggests that the higher exploration-exploitation balance provided by the lower epsilon decay rate might have allowed the agent trained with an epsilon decay of 0.7 to discover unique strategies or exploit certain game dynamics, leading to sporadic but impactful bursts of success.

## VI. Agent Performance

All results of the agent's performance that were displayed in this report were done using randomized seeds. At first, the score achieved by the agent under all the different scenarios is purely random as it is fully exploring and not exploiting. However, as the number of episodes grows, the agent learns that heading directly towards a corner of the map allows it to 'live' for longer periods of time. That is because the way that its reward system is set is that it gains 10 points for each pellet it eats, and loses 100 points for every life it loses. The agent learns that it is better to stay away from the ghosts and that being in the map's corners allows it to generally be far from them all. To retrieve the actual score that each agent reached per episode, simply add 300 to the value of each graph as these scores account for the -300 points due to losing all of the agent's lives before a game can be ended.

## VII. Running The Script

To run this script, a README file is provided which you can refer to. The executable can be run with different hyperparameters, but it is recommended to only change the 'display' and 'learning' arguments with display being true and learning being false by default which allows the script to visually display the agent's behaviour. Finally, the 'weights path' argument can be changed to any of the pre-trained weights for the various training conditions that were highlighted in this report to see their individual performances.

## References

[1] Volodymyr Mnih et al. "Playing Atari with Deep Reinforcement Learning". In: *arXiv preprint arXiv:1312.5602* (2013). URL: https://arxiv.org/pdf/1312.5602.pdf.