Design and Implementation of a Reinforcement Learning Agent for Pacman Gameplay

Hazhir Amiri
Fachhochschule Dortmund
Dortmund, Germany
hazhir.amiri001@stud.fh-dortmund.de

Raed Kayali
Fachhochschule Dortmund
Dortmund, Germany
raed.kayali001@stud.fh-dortmund.de

I. Introduction

Reinforcement learning has paved the way for enhancing agents abilities to propagate through a complex unknown environment with interaction. The RL agent learns about the environment based on rewards. Positive points are given when the agents makes a correct move and a negative point or a deduction is a result of a failure. The agent goes through trial and error sequences guided by feedback mechanisms for the actions. This makes RL suited for overcoming obstacles that involves planning, optimization and adaptation . The main goal of the semester project is the utilization of RL in creating a Pacman agent that is capable of solving problems which are based on ghosts and gaining rewards as points or cookies collected making Pacman an excellent testing ground for RL due to it having a structured but at the same time variable gameplay mechanics that involves navigation through mazes, points collection, avoiding enemies. The core of this project involves the design, implementation, and evaluation of reinforcement learning agents tailored for multiple Pacman levels.

II. Methodology

A. Deep Q-Network (DQN)

The Deep Q-Network (DQN) is a reinforcement learning algorithm that combines Q-learning with neural networks. this aids the agents in navigating through complex environments that has multiple states and action combinations. Instead of using tables like done in Q-learning. DQN utilizes neuralnetworks to approximate values that gives weight for each possible action making it a process with high scalability and a more compelling approach towards a high dimensional task like Pacman

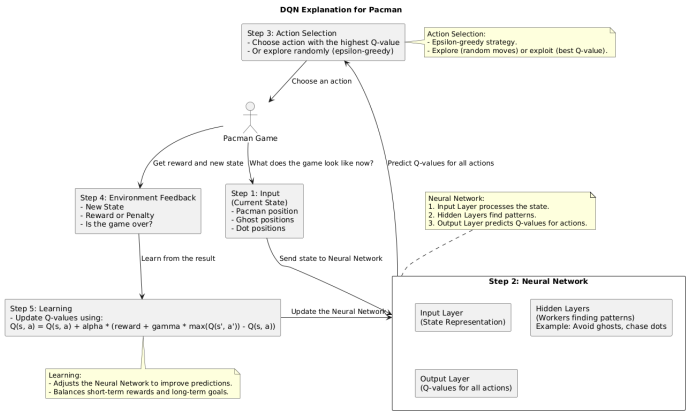

Fig. 1. DQN Architecture in Pacman game

B. How DQN works

1) Input layer: This layer contains the current state of the game as an array of numerical values that corresponds to positions of Pacman, dots, and ghosts that are then fed into the neural network.

2) Hidden layer: This layer consists of 2 connected layer with 128 neurons each and a ReLU activation function. This aids the agent in creating a pattern to reach maximum rewards by collecting cookies and avoiding ghosts

3) Output layer: This layer gives values to each possible action that can be taken by the agent. The action wit the highest value is then selected.

C. Action Selection

epsilon-greedy strategy is used in the selection based on Exploration, a random choice and Ex-

ploitation, a choice based on value received from neural network prediction .

D. Learning Process

After taking the action, feedback is given to update the neural network using the following formula:

$$Q(s,a) = Q(s,a) + \alpha \left(r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right) \quad (1)$$

Where α is the Learning rate and γ is a Discount factor for balancing short-term and long-term rewards.

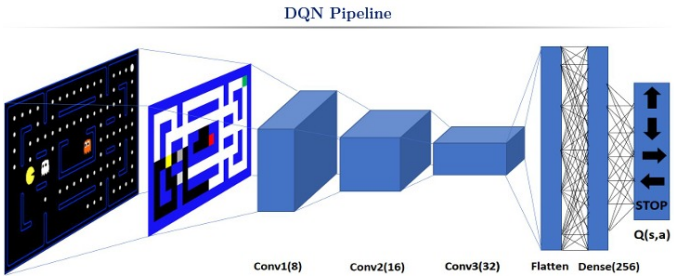

Fig. 2. DQN Architecture

E. Q-Learning and Dyna-Q

In order to train an agent for the Pacman game, two methods were considered: Q-Learning and Dyna-Q as choices from model-free and model-based methods.

1) Q-Learning: Q-Learning is a well-known reinforcement learning method within the data science committee and its simplicity is a perfect fit for the deterministic world of Pacman. In Q-Learning, the agent interacts with the world over many episodes (or epochs) to build a Q-table that represents the best action for any given state. As the agent starts interacting with the world, it can either explore or exploit. To explore the agent takes a random action within the environment and observes its result to update the q-table. Within the random actions, the agent can either consider all possible actions or filter the actions to ones that could be more useful. On the other hand, the agent can choose its next action through exploitation. This means that the agent will utilize its pre-existing knowledge (values in the q-table) to derive its next action. The decision between exploration and exploitation can be made through various methods. One of the most popular of those is called ϵ-greedy, where an ϵ value is defined, and at each turn, a random number usually from 0 to 1 is chosen and if the number is smaller the ϵ, the agent explores the environment. Besides ϵ-greedy, other methods such as a decaying ϵ can be utilized to reduce random actions over time in favor of maximizing rewards. After the agent takes an action, it observes the environment for the result. From this result, the agent receives a reward. To train a proper agent, a proper reward structure needs to be defined that will reward the agent for the proper actions. Besides being rewarded for winning the game or penalized for losing it, the agent, for example, could be rewarded for eating a dot (which in turn will help achieve the end goal of winning the game). After the reward for the action has been determined, the reward will be saved for the corresponding state and action pair. Over time, the agent will update these rewards and explore more states. At the end during evaluation, the agent can try to select the action with the highest rewards for all states it has encountered.

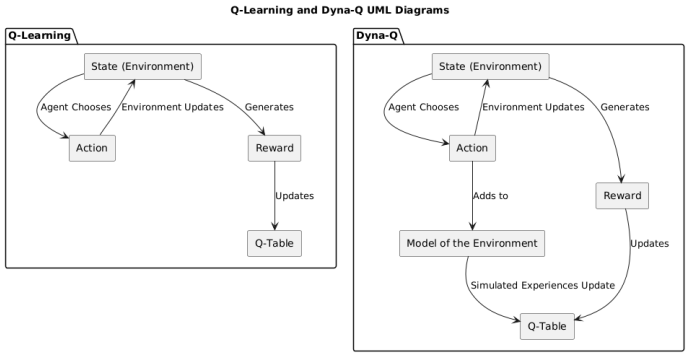

Fig. 3. DQN Architecture

2) Dyna-Q: Dyna-Q is an extension of the Q-Learning method where a simple model is added to Q-Learning so that the agent can attain a better understanding of its world and construct a better q-table. Dyna-Q allows the agent to learn from the environment without actually interacting with it. To achieve this, a model of the environment is built that will help update the values of the q-table. After each real interaction with the environment, the agent considers the states it has already observed and randomly chooses a state-action pair from them. The agent repeats this step arbitrarily and then updates the q-table using the rewards it has already recorded for the state-action pairs. This helps the agent ex-

perience the world without having to interact with it.

III. Experiments

Experiments were conducted to train the reinforcement learning agent across various Pacman levels. The training involved analyzing the impact of key parameters like learning rate, discount factor, and epsilon decay on the agent's performance.

A. Actor-Critic Algorithm

This algorithm utilizes 2 components to achieve its goal. First is the agent who is responsible for making a choice of which path or action to take based on a Softmax Activation Function. The other is the critic responsible for evaluating the current state's quality based on the current state values. This helps in providing feedback on the action taken by the actor in order to improve the decision taken in the next interval.

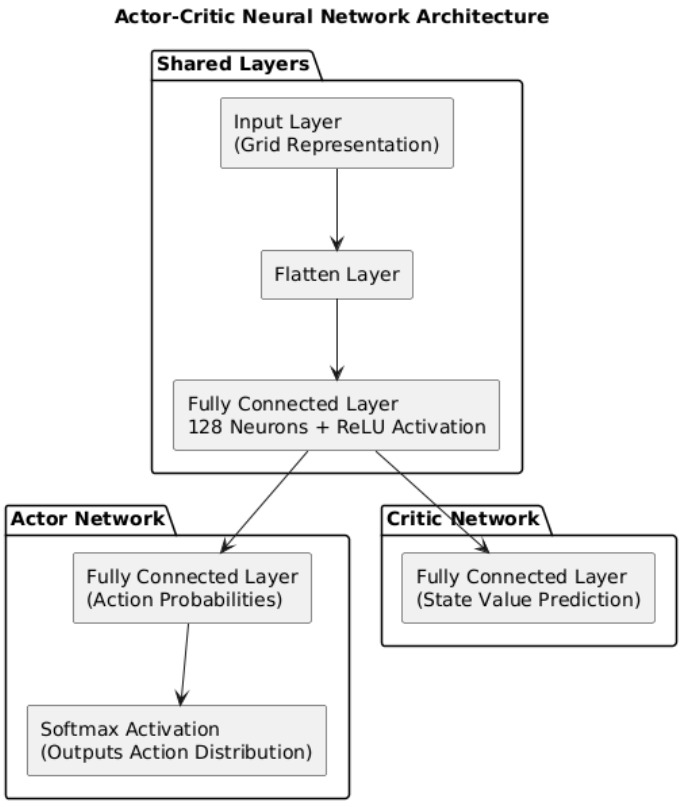

Fig. 4. Resnet34 Architecture

B. Architecture

1) Shared layers: This layer is composed of an input layer containing the grid representation of the environment of the Pacman game, a flatten layer, and a fully connected layer with a ReLU activation function.

2) Actor: Features are passed to this layer as an input into another fully connected layer. The output here is the probability distribution of the actions possible to be taken by the agent, which in our case are the 4 possible directions the Pacman can move to. This is done using a SoftMax function which is basically a function that takes raw numerical values and outputs probabilities.

3) Critic: A fully connected layer receives the same features passed to the actor. However, the action done here is different. The goal is to evaluate the policy by predicting the state value which helps the actor improve its decisions.

C. Observed Issues

1) Loss Value Fluctuations: During the training of the model, a high change in the loss value was observed as the value kept increasing and decreasing without a meaningful pattern. This could have occurred due to an imbalance within the randomness of the choices and the choices taken based on the learning based on feedback. Moreover, the gradient clipping done might have not been effective.

This resulted in an agent that could not navigate the maze effectively and made poor choices in the direction choices.

IV. Results

1) Q-Learning and Dyna-Q: The most important aspects to consider within Q-Learning and Dyna-Q are the hyperparameters that affect the exploration strategy and the resulting q-values after each step. To find out the best possible combinations for each level of the game, a hyperparameter matrix script was implemented that checks different hyperparameter values together and saves the result as plots for observation. For example in the following figure, the increase in discount factor leads to a loss in cumulative rewards.

It is also apparent that a lower learning rate (represented by LR) yields higher rewards. Note that 0 simulated space essentially depicts a Q-Learning

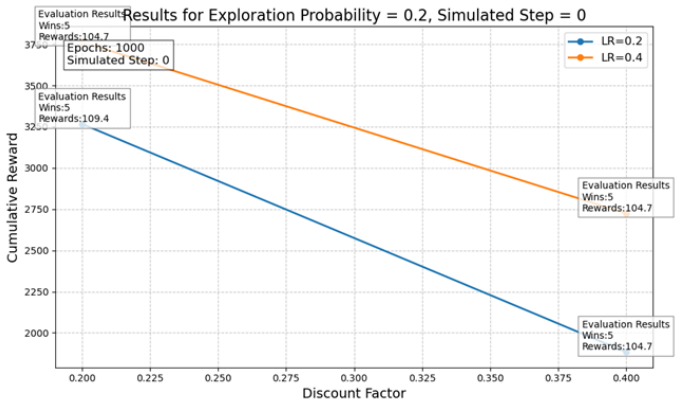

Fig. 5. Dyna-Q hyperparameters rewards for level 4

agent and any higher value is a Dyna-Q agent. To choose the most suitable hyperparemeters, the culminative reward values were the most important criteria. Furthermore the performance during evaluation (represented by wins and rewards within the evaluation results box) were also considered.

2) DQN: The DQN model had a promising results. The agent was able to navigate towards the cookies and away from the ghosts at the start of the run showcasing its ability to learn fundamental strategies and adapt to the environment. However, it started to lose navigation towards the goal. Consequently, the agent's performance deteriorated, and it frequently failed to complete the game. This could be tremendously improved through tuning the hyperparameters more and increasing the number of episodes.

V. Conclusion

This paper explored the utilization of reinforcement learning algorithms, including Q-Learning, Dyna-Q, Deep Q-Networks (DQN) and Actor-Critic algorithms in training a pacman agent to be able to navigate throughout the environment in a way that would increase the rewards in collecting cookies and reduce the failures in being captured by ghosts

The experiments demonstrated that while Q-Learning is a simple and an effective baseline for the use of reinforcement learning , Dyna-Q offered an enhanced learning capability for the agent by simulating more experiences without direct interaction with the environment.

The DQN approach highlighted the significance of utilizing neural network-based reinforcement learning as Initial results proved an agent's ability to adapt effectively and the possibility for improvement in future work

the Actor-Critic Model did not perform in a correct manner and gave results of no rewards gained when tested on the models expected due to the oscillating loss within the training

References

References

[1] Sutton, R. S., Barto, A. G., *Reinforcement Learning: An Introduction*, MIT Press, 2018.

[2] Abeynaya Gnanasekaran, Jordi Feliu Faba, and Jing An, "Reinforcement Learning in Pacman," *SUNet IDs: abeynaya, jfeliu, jingan*.

[3] Mnih, V., et al., "Playing Atari with Deep Reinforcement Learning," arXiv preprint arXiv:1312.5602, 2013.

[4] Pacman environment, *Gym Pacman*, OpenAI, https://gym.openai.com/envs/Pacman/.

[5] Kuo Li, Qing-Shan Jia, and Jiaqi Yan, "An Actor-Critic Method for Simulation-Based Optimization," *IFAC-PapersOnLine*, vol. 55, no. 11, pp. 7–12, 2022. IFAC Workshop on Control for Smart Cities CSC 2022. DOI: .

Affidavit

We, Raed Kayali, Hazhir Amirir herewith declare that we have composed the present paper and work by ourselves and without use of any other than the cited sources and aids. Sentences or parts of sentences quoted literally are marked as such; other references with regard to the statement and scope are indicated by full details of the publications concerned. The paper and work in the same or similar form has not been submitted to any examination body and has not been published. This paper was not yet, even in part, used in another examination or as a course performance.

Dortmund 17th January 2025