GYMMASTER

A REINFORCEMENT LEARNING PROJECT

A black and white logo

Description automatically generated

AUTHORS: VYOMA MANKAD

GAURAV TATAPATE

ANSH SEMWAL

MENTORS: ADITYA VIVEKANAND

ATHARV DUBEY

INDEX

|  |  |  |
| --- | --- | --- |
| Sr No | Content | Pg No |
| 1 | Overview |  |
| 2 | Introduction:   * What is RL? * Features of RL * Applications of RL |  |
| 3 | Basic Concepts   * K armed bandit problem * Markov Decision process * Dynamic programming * Gridworld Problem |  |
| 4 | Deeper Concepts   * Monte Carlo and blackjack * What is Q learning? * Mountain Car Problem * Cartpole Problem * Frozen Lake Problem |  |
| 5 | DQNS:   * What are DQNS? * Features of neural network * Pong implementation using CNNS |  |
| 6 | * Problems faced * Future tasks * References |  |

**Overview**

Our project aims to understand the various concepts involved in the domain of Reinforcement Learning. It involves learning and implementing classic reinforcement learning problems to forge a better understanding of the domain. We aim to finally implement a basic pong Atari game using basic NumPy functions.

A drawing of a stick in a square box

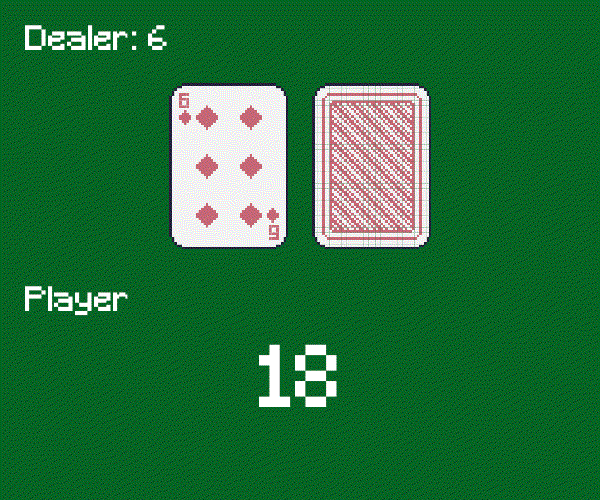
Description automatically generated A black line on a white background

Description automatically generated

A screenshot of a video game

Description automatically generated A screenshot of a computer

Description automatically generated



**REINFORCEMENT LEARNING**

**What is Reinforcement Learning?**

A person standing next to a robot

Description automatically generated

Reinforcement Learning is a type of machine learning.  It is about taking suitable action to maximize reward in a particular situation.

It is employed by various software and machines to find the best possible behaviour or path it should take in a specific situation.

Reinforcement learning differs from supervised learning in a way that in supervised learning the training data has the answer key with it, so the model is trained with the correct answer itself whereas in reinforcement learning, there is no answer, but the reinforcement agent decides what to do to perform the given task. In the absence of a training dataset, it is bound to learn from its experience.

A diagram of a diagram of a brain

Description automatically generated with medium confidence

**Parts of Reinforcement Learning:**

**States:** A *state* represents the current situation or configuration of the environment. It encapsulates all the information the agent needs to make decisions. For example, in a game like chess, a state would include the positions of all pieces on the board.

**Actions**: *Actions* are the possible moves or decisions an agent can make when in a particular state. In each state, the agent can choose an action from a set of allowed actions that will influence the environment. For example, in a self-driving car simulation, an action could be to accelerate, brake, or turn.

**Environment**: The environment is the external system that the agent interacts with. It takes the agent's actions and transitions to a new state while providing a reward or penalty based on the action. For instance, in a robotics task, the environment includes the physical world the robot operates in, such as the objects it manipulates and the space it moves through.

A diagram of a diagram of a state and reward actions

Description automatically generated

**Basics:**

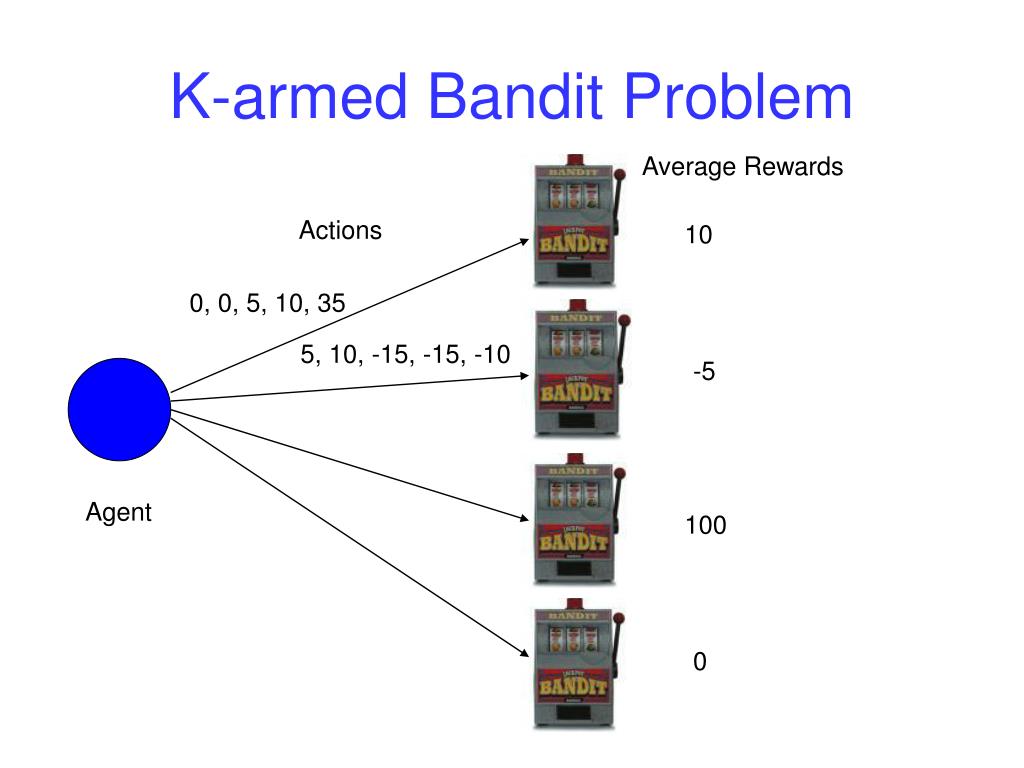
**K armed Bandits**

The Multi-Armed Bandit (MAB) problem is a classic problem in probability theory and decision-making that captures the essence of balancing exploration and exploitation. This problem is named after the scenario of a gambler facing multiple slot machines (bandits) and needing to determine which machine to play to maximize their rewards.



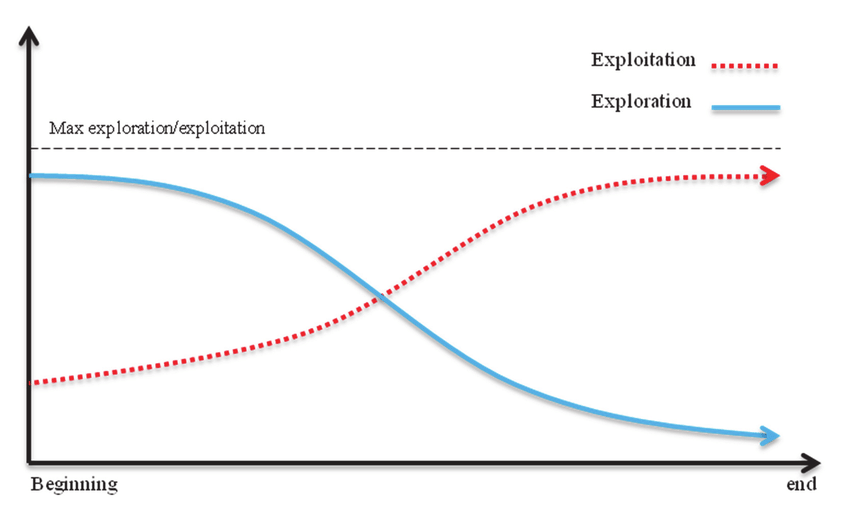
In the Multi-Armed Bandit problem, an agent is presented with multiple options (arms), each providing a reward drawn from an unknown probability distribution. The agent aims to maximize the cumulative reward over a series of trials. The challenge lies in choosing the best arm to pull, balancing the need to explore different arms to learn about their reward distributions and exploiting the known arms that have provided high rewards.

* ***Arms****:* K independent arms, each with an unknown reward distribution.
* ***Rewards****:* Each arm i provides a reward Ri, drawn from an unknown distribution with an expected value​​.
* ***Objective****:* Maximize the cumulative reward over T trials.



The central dilemma in the MAB problem is the trade-off between exploration (trying different arms to gather information about their rewards) and exploitation (choosing the arm that has provided the highest rewards based on current information). Balancing these two aspects is crucial for optimizing long-term rewards.

**What is exploration and exploitation?**



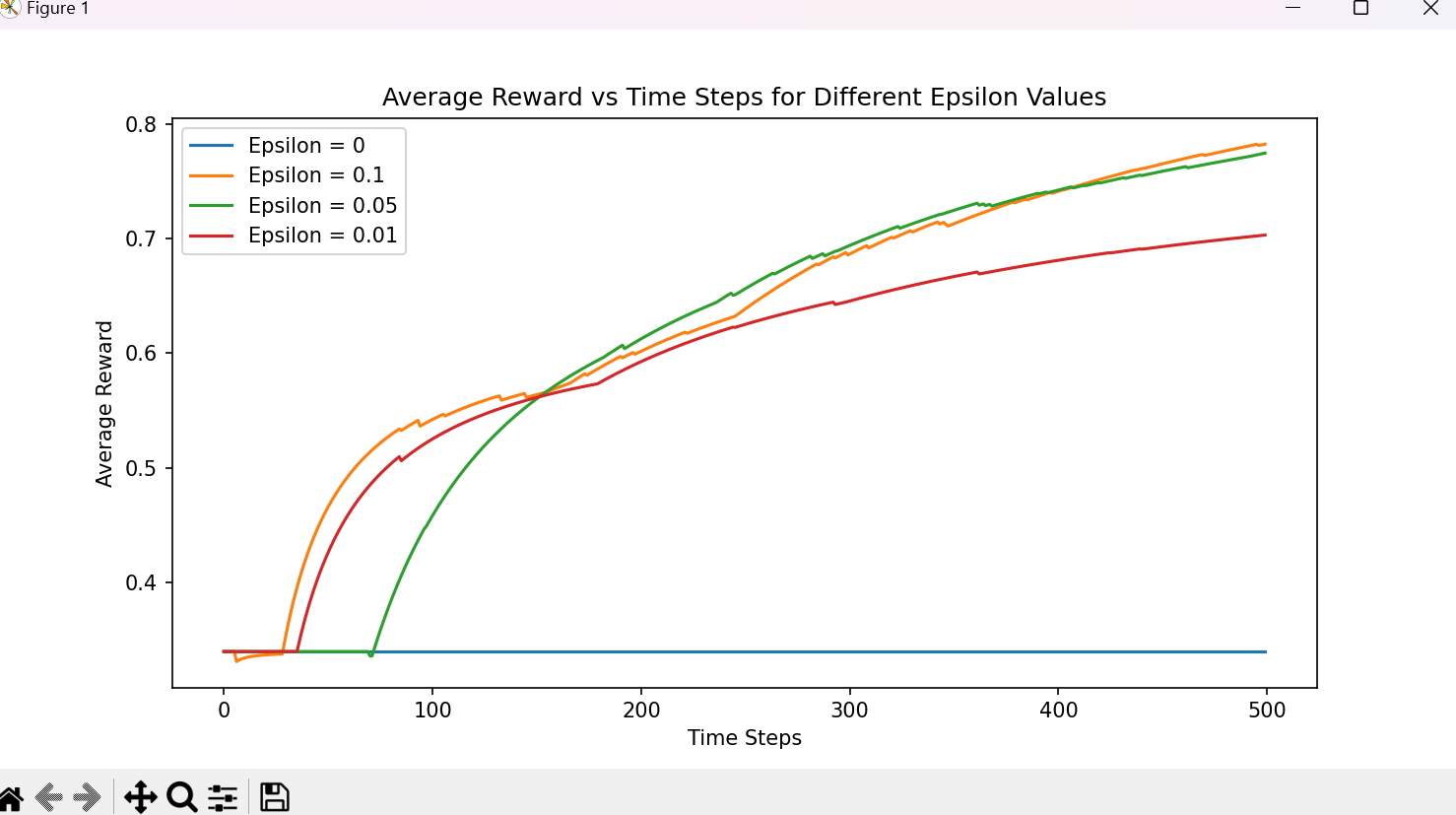
***Exploration:*** It is the process by which a reinforcement learning agent tries out different actions to discover which ones yield the best rewards, helping it learn more about its environment.

***Exploitation:*** It is the process in reinforcement learning where an agent selects actions that it believes will yield the highest reward based on its current knowledge, rather than trying new actions.

Solving exploration-exploitation trade-off:

**Epsilon Greedy Strategy:**

The epsilon-greedy strategy is a method in reinforcement learning where an agent chooses to explore random actions with a probability of ε (epsilon) and exploits the best-known action with a probability of 1-ε. This balances exploration and exploitation.



**Markov Decision Process**

A Markov Decision Process (MDP) is a mathematical framework used to model decision-making in environments where outcomes are partly random and partly under the control of a decision-maker.

1. **Markov Property**

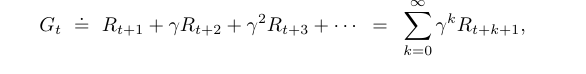
The Markov property states that the future state of a process depends only on the current state and action, not on the sequence of events that preceded it.

A math equation with black text

Description automatically generated

1. **Returns:**

Theyrefer to the total accumulated reward an agent receives over time, typically starting from a specific state or action. It can be calculated as the sum of rewards obtained in future steps, often discounted by a factor to prioritize immediate rewards over distant ones.



1. **Policy**

Policy in reinforcement learning is a strategy or rule that dictates the action an agent should take in each state of the environment. It can be:

* ***Deterministic:*** Directly maps each state to a specific action.
* ***Stochastic:*** Provides a probability distribution over possible actions for each state.

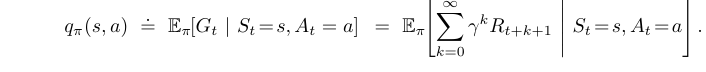
1. **Value function**

The state value function tells us the expected cumulative reward that an agent can obtain starting from a state s and following policy pi



1. **Action value function**

The action value function estimates the action that a particular policy takes and the expected cumulative reward that an agent can achieve

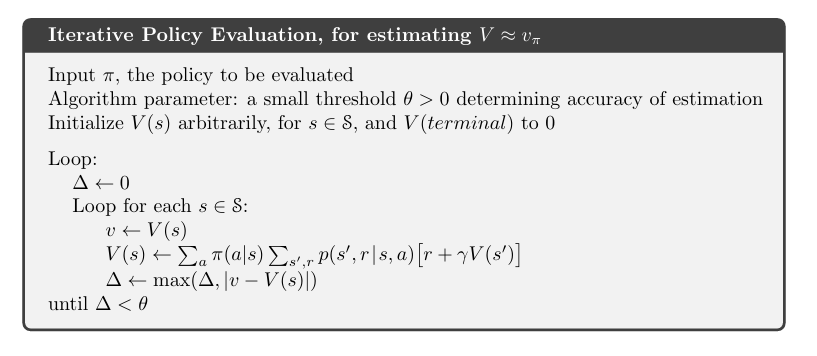


**Dynamic Programming:**

The term dynamic programming (DP) refers to a collection of algorithms that can be used to compute optimal policies given a perfect model of the environment as a Markov decision process

* 1. **Policy evaluation**:

It is the process of calculating the value function for a given policy in a Markov Decision Process (MDP). It determines the expected cumulative reward for each state under the current policy, providing a measure of how good the policy is in terms of the rewards it will yield over time.



* 1. **Policy improvement**

Policy improvement is the process in reinforcement learning where a given policy is refined by selecting actions that lead to higher expected returns based on the current value function. During this step, the agent updates its policy by choosing the best possible actions in each state, thereby creating a new, improved policy.

* 1. **Value iteration**

Value iteration is a dynamic programming algorithm used to find the optimal policy in a Markov Decision Process (MDP). It combines policy evaluation and policy improvement into a single step by iteratively updating the value function for each state.

A screenshot of a math problem

Description automatically generated

**GRIDWORLD:**

A screenshot of a math game

Description automatically generated

Aim: The aim of the gridworld problem is to find the optimal policy that will yield the maximum cumulative reward. It involves obtaining the best value function for a state and accordingly taking further actions

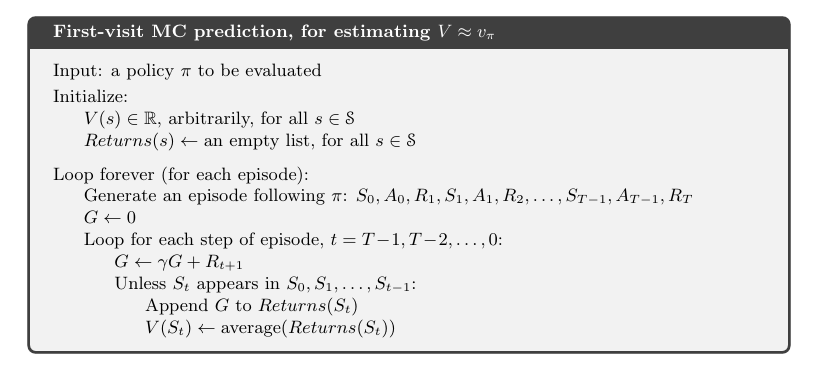
Features:

* The environment is represented as a grid of cells (e.g., 4x4, 5x5).
* Each cell in the grid represents a state.
* The agent can typically move in four directions: up, down, left, or right
* Cells may have positive rewards (e.g., goal cell) or negative rewards (e.g., pit).
* Some cells may be terminal states where the episode ends (e.g., goal or pit).
* There is a value function which estimates expected cumulative reward of each state
* Discount factor tells us the importance of future rewards as compared to immediate rewards.
* We use concepts of dynamic programming and mdps to implement gridworld

**Monte Carlo Methods:**

Monte Carlo methods estimate value functions and optimize policies by averaging the returns from sampled episodes of interaction with the environment.

* They directly learn from episodes
* They update value estimate after end of episode
* The value function is updated according to the empirical mean
* Show high variance and zero bias
* Works only for complete episodes



**Blackjack Problem**

A screenshot of a game

Description automatically generated

Blackjack is a game where players aim to get a hand value close to 21 without going over, while competing against the dealer. Players make decisions on whether to hit, stick, double down, or split based on their hand and the dealer’s up card. The dealer follows a set strategy to complete their hand, and the outcome is determined based on the final hand values.

The goal is to accumulate a hand value as close to 21 as possible without exceeding it, and to have a higher value than the dealer’s hand.

The state of the game is represented by:

* The player's current hand value.
* The dealer’s visible card.
* The player's current total of aces and cards (in some formulations).

The possible actions the player can take are:

* ***Hit:*** Draw another card from the deck.
* ***Stick (Stand):*** End the player's turn and keep the current hand.
* ***Winning:*** Positive reward if the player’s hand value is closer to 21 than the dealer’s hand or if the dealer busts.
* ***Losing:***Negative reward if the player’s hand value exceeds 21 (busts) or is lower than the dealer’s hand without busting.
* ***Push (Tie):*** Neutral reward if the player’s hand value equals the dealer’s hand value.

Monte Carlo methods in Blackjack involve simulating multiple games to estimate the value of state-action pairs and improve the player's policy. By averaging the returns from these simulations, the player’s strategy is refined to maximize expected rewards, ultimately leading to an optimal policy for playing Blackjack.

Q LEARNING IMPLEMENTATIONS

**What is Q learning?**

Q-learning is an off-policy algorithm that aims to learn the value of state-action pairs to determine the best policy for maximizing cumulative rewards in each environment. It updates the Q-values (expected future rewards) based on the agent's experiences to improve decision-making.

A math equations on a white background

Description automatically generated

**Why Q learning?**

Q-learning is used because it provides a model-free, simple, and effective way to learn optimal policies in reinforcement learning tasks. It allows agents to learn from experience without needing a model of the environment, handles stochastic environments, and is applicable to various types of problems.

Implementations:

1. **Mountain car**

A black and white line

Description automatically generated

The aim of this problem is to make a car stuck in a valley gain enough momentum to reach the flagpost.

The state space consists of:

* **Position:** The car's position along the track.
* **Velocity:** The car's current speed (which can be positive or negative).

The actions available are:

* **Accelerate left:** Move the car left.
* **Accelerate right:** Move the car right.
* **No action:** Do nothing.

**Rewards:**

* **-1:** At each time step, a small negative reward is given to encourage the car to reach the goal quickly.
* **0:** The episode ends when the car reaches the top of the right hill, and no additional reward is given for reaching the goal.

1. **Cartpole Problem**

A diagram of a pivot point

Description automatically generated

Aim: The agent must learn to apply forces to the cart to keep the pole balanced upright for as long as possible.

States:

Cart position

Cart velocity

Pole angle

Angular velocity

Actions:

Move left

Move right

Reward:

+1 reward for balancing

-1 reward for other cases



* 1. **Frozen lake**

In the Frozen Lake environment, you control an agent that must navigate from a starting point to a goal across a grid representing a frozen lake. Some cells in the grid are slippery or unsafe, and stepping on these cells results in falling into a hole or slipping, which impacts the agent's progress.

**S:** Starting point

**G:** Goal

**F:** Frozen (safe)

**H:** Hole (unsafe)

The actions available are:

* **Left**
* **Right**
* **Up**
* **Down**

The rewards are:

* **+1:** Reaching the goal state.
* **0:** Stepping on a frozen cell.
* **-1:** Falling into a hole.

All these implementations update q values for actions based on their states and accordingly find an optimal policy to follow

**DEEP Q LEARNING**

Deep Q-Learning (DQL) is an extension of Q-learning that uses deep neural networks to approximate the Q-value function, enabling it to handle environments with large or continuous state spaces where traditional Q-learning would be impractical

What are Neural networks?

A diagram of a network

Description automatically generated

***Neurons:*** Basic units of a neural network, analogous to biological neurons. Each neuron receives inputs, processes them using an activation function, and passes the output to other neurons.

***Layers:*** Neural networks consist of layers of neurons:

* ***Input Layer:*** Receives the raw data.
* ***Hidden Layers:*** Intermediate layers that process data through learned weights and activation functions.
* ***Output Layer:*** Produces the final prediction or classification.

***Weights and Biases:*** Parameters that the network learns during training. Weights adjust the importance of inputs, and biases help shift the activation function.

***Activation Functions:*** Functions applied to the output of each neuron to introduce non-linearity. Examples include sigmoid, ReLU,tanh

*Training:* The process of adjusting weights and biases to minimize the difference between the predicted output and the actual output using optimization algorithms (e.g., Gradient Descent).

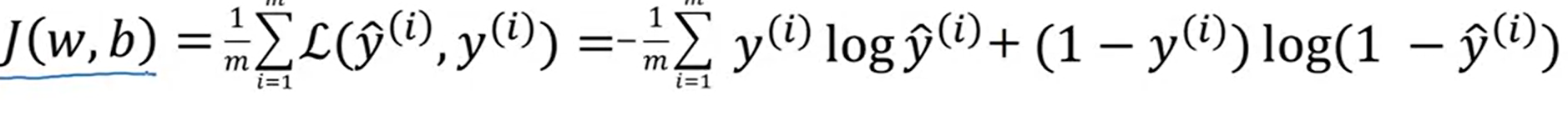
***Backpropagation:*** An algorithm used during training to compute gradients of the loss function with respect to each weight by propagating errors backward through the network.

***Loss Function:*** Measures the error between a model's prediction and the actual target for a single data point.

A black and white background with black text

Description automatically generated with medium confidence

***Cost Function:*** Aggregates the loss across all data points and may include additional terms (like regularization) to guide model training.



**Pong implementation**

In the Atari Pong implementation, we have made use of CNN networks hence creating a deep learning model which we train and implement using python and NumPy

To implement Pong using CNNs:

1. **Initialize Environment:** Use OpenAI Gym's Pong environment.
2. **Build CNN Model:** Create a neural network with convolutional layers to process image frames and predict Q-values.
3. **Train with DQN:** Use Deep Q-Learning, which combines CNNs for feature extraction with Q-learning for decision-making. Incorporate experience replay and a target network for stability.
4. **Action Selection:** Use the CNN to choose actions based on predicted Q-values and update the model based on rewards.

Why CNN?

Diagram of a diagram of a computer network

Description automatically generated with medium confidence

**CNNs** are used with Pong because they efficiently process high-dimensional image data (e.g., raw pixel data) and automatically extract relevant features. This allows them to handle complex environments better than traditional Q-learning, which struggles with large or continuous state spaces. CNNs, combined with algorithms like Deep Q-Learning (DQN), improve performance by learning from visual input and making more informed decisions.

Problems faced:

* Gym version not compatible with pcs
* Dependency errors arise while trying to execute the pong game
* We can’t train the pong code due to these

Future Goals

* Implementing pong using libraries such as keras and TensorFlow for better and efficient results
* Implementation of RL on a self-balancing bot
* Usage in simulation of bots on platforms like gazebo
* Developing a RL based drone to deliver packages

References

* RL by Barto and Sutton
* Lecture series on RL by David Silver
* Course on Neural Networks by Andrew Ng